

# Group Report on Final Software System

Group 6

Client: Matt Morgan

Supervisor: Louise Knight

Members:

Abdu Dihan (c1301922)

Andrei Hodorog (c1332008)

Ben Leonard-Legarde (c1322567)

Karl Latham (c1335031)

Rhian Milcoy (c1305879)

Robert Horton (c1323141)

Roberto Dyke (c1317850)

Sam Maltby (c1334821)

Zain Tahir (c1308094)

## Table of Contents

### [Introduction](#)

#### [Report contents](#)

#### [Competition](#)

#### [Solution Functionality](#)

### [Progression](#)

#### [Functional Components](#)

##### [User End \(Front\)](#)

###### [Login screen](#)

###### [Main screen](#)

###### [Results screen](#)

###### [Overall](#)

##### [Developer End \(Back\)](#)

###### [Application Authentication](#)

###### [Data Volumes](#)

###### [Facebook Query Limits](#)

###### [Server Requirements](#)

###### [pThreads Test Server](#)

###### [System Compilation](#)

#### [Team Delegation & Structure](#)

##### [Modular Code Design & Its Impact On Group Work](#)

#### [Client Meetings](#)

### [Overall Software System](#)

#### [Changes to Design](#)

#### [AJAX](#)

#### [Networking](#)

#### [Calculation Process](#)

##### [Variables](#)

###### [Variable Contingency](#)

###### [Missing Functions](#)

##### [Evaluate Friends](#)

##### [Profile User](#)

##### [Offset](#)

#### [Admin Section](#)

##### [Settings File](#)

### [Testing](#)

#### [Developers Perspective](#)

##### [Back End](#)

###### [API-to-Algorithm Conversion](#)

###### [Algorithm](#)

###### [Display Information](#)

##### [Functions](#)

[User's Perspective](#)[Test Cases](#)[Quality of code](#)[Justification/Evaluation](#)[Justification for design choices](#)[Visibility](#)[Feedback](#)[Constraint](#)[Mapping](#)[Consistency](#)[Affordance](#)[Predictability](#)[Synthesizability](#)[Familiarity](#)[Justification for implementation choices](#)[Assumptions Log](#)[Software strengths & limitations](#)[Requirements](#)[MoSCoW](#)[Must Have](#)[Should Have](#)[Could Have](#)[Success Criteria](#)[User](#)[See who my closest friends are across multiple social media sites](#)[See how close I am with certain friends](#)[See how many connections I have](#)[Choose which social media sites I use](#)[Grant permission](#)[Have the ability to manually change the results](#)[Share the results with friends](#)[See where the people are who I interact with](#)[Admin](#)[Fix bugs](#)[Add new features](#)[Monitor the consumer base](#)[Business Plan](#)[Issues Considered](#)[Legal](#)[Code Ownership](#)[Software Licensing](#)[Facebook Platform Policy](#)[Data Protection](#)

- [Ethical](#)
- [Social](#)
- [Professional Issues](#)
- [Conclusion](#)
- [Future Work](#)
- [References](#)
- [Appended Zip File Information](#)
- [Appendix](#)
  - [Timings](#)
  - [Implementation Work](#)
  - [Team Organisation](#)
  - [Documentation Work](#)
- [Gantt Chart](#)
  - [Part 1](#)
  - [Part 2](#)
- [Class Diagram](#)
  - [Part 1](#)
  - [Part 2](#)

## Introduction

### Report contents

This report marks the end of the implementation in the development of the project. The report will include information on the progression since the last report, including development in the design stage, functional components, and both user-end and developer-end progression.

Some work from our last report will be referred to, this will mostly be included to support decisions we have made and to show changes we have made to better the end product.

In this report there shall also be a section included on team progression, in addition to this a Gantt chart will be included to show specifically the different tasks that each member of the group have carried out throughout the duration of the project.

Since the design report there has been a few changes to the design, these modifications shall be specified and justified later in the report.

For now, changes that have occurred since the last report include:

- Class structure - altered
- Facebook getter class - partially implemented
- Admin section - not implemented
- Friendship variables - partially implemented & partially modified

### Competition

Some applications that are similar to what we have create include: MyTopFans Pro- which is available on Apple's iTunes store at a cost of £1.99. The description of the app provides information on how it analyses your Facebook profile, and reveals the identity of your 'fans' and their level of interest in you. Features of the application include the option to filter fans, based on Gender, Relationship status, or friend status. A 'Position Tracker' which helps users to track the movement of the users top 20 fans, and a 'Fanbar' to display how much users 'care' about you. The application also displays photos of your fans, and their latest photos. The application so have has received one 5 star review. Research also revealed another application that also applies to the analysis of Facebook friends, but is used for: 'Revealing which of your Facebook friends make you happier'. 'The application takes a few seconds to analyse the posting behavior of the user's top 25 Facebook friends, ignoring anyone who has written fewer than 10 status updates in the past year or whose privacy settings hide their posts from apps.'

This application however focuses more on the linguistics used on facebook, but also measures the interaction between users in the same way as our application does i.e. through

tagged photos, wall posts, and other interactions over the social media site. Application for other social media sites were slightly more challenging to research, with no downloadable applications available for users to use, but instead analytical tools such as- The Archivist, SocialBro and MentionMap. These tools then help build up a rich picture of who the top twitter users are, tweet vs. retweet ratio, and sources of which tweets are sent from, just to name a few.<sup>[4]</sup>

## **Solution Functionality**

In terms of the initial brief set at the beginning of the school year we have built a social media friend mapper. The application uses the Facebook API to identify the strength of the connections between the user and their friends. By looking at a user's interaction, personality, and physical environment we have been able to approximate a friendship value. We have been able to take the values procured and display them on a diagram that depicts the 'inner circle' and 'outer circle' for a group of friends.

## **Progression**

The work undertaken during this part of the project was broadly split into two aspects, following what had been mapped out in the design section. That being the coding of the system itself, and the documentation for the system, with the business idea alongside it.

Front end development concentrated on creating an aesthetically pleasing and functioning user interface for the system. Back end development focussed on obtaining the data required by the algorithm and the algorithm itself.

As part of this project the team was split into different groups in order to maximise efficiency and to work to individuals strengths. One member of the group took on the role of project manager, and they oversaw all other members of the group in completing their delegated sections of work. This allowed a central point of contact for anyone who was unsure of what they needed to do, or who had problems with part of the work. This was useful for all members of the group.

Two members of the group took on the role of primarily writing up reports and documentation, as well as dealing with designing and writing a business plan for the application. They were supervised throughout by the project manager.

The remaining six members of the group took on developing the application, and further split themselves into two groups, two as back end developers, two as middle end developers, and one as front end developer. Work was handed to each developer on a week by week basis by the project manager, laying out a specification of what was needed. The project manager towards the applications completion then coordinated the application "being stitched together" into a functioning system.

Group meetings were held every week to discuss the progress made towards the solution. This was a useful time to discuss problems related to the tasks delegated to an individual or the project in general, this opportunity was seized to improve our user interface as a group, as

well as from external feedback. Often weekly tasks that hadn't been assigned would be discussed between those present and then assigned to the most suited team member.

### Gantt Chart

Specified in the feedback from the last report, the client notified us that our Gantt chart was slightly unclear. Included in this Chart, is a full specification of the tasks of every group member, and the time scale required for each task. The Gantt chart again was completed using a specific online tool to help us manage our time during the implementation stage of the project.<sup>[6]</sup>

Project stages		Status
<b>Meetings</b>	<b>Name of Person Responsible</b>	▼
Group Meetings	Roberto, Rhian	Complete
Client Meetings	Rhian	Complete
<b>Group Report</b>	<b>Name of Person Responsible</b>	▼
Introduction	Ben, Rhian	Complete
Progression	Andrei, Roberto, Karl	Complete
Testing	Zain, Andrei	Complete
Justification/Evaluation	Roberto, Andrei, Karl, Roberto	Complete
MoSCow Requirements	Rob H, Karl	Complete
Functional Components	Roberto, Ben, Karl, Rob H, Andrei, Zain	Complete
Legal/Social/Ethical Issues	Sam	Complete
Conclusion	Roberto	Complete
Structure/Write Up of Rep...	Sam and Rhian	Complete
<b>Implementation</b>	<b>Name</b>	▼
Font End	Andrei	Complete
Back End	Roberto, Karl, Ben	Complete
Write up of Functions	Ben, Roberto, Rob H, Andrei, Abdu, Karl, Zain	Complete
Delivery/Presentation	All Group Members	Complete

The Gantt chart (see appendix) is set from the period of the 8th of March (where the previous Gantt chart was left) until the end of the project. As you can see most of the work was split, with the implementation stage mostly being completed before Easter, and the report writing being completed over the Easter break. Some tasks were stretched over a number of weeks, such as the writing of code, i.e. Group members were given a different function each week to complete over a period of 4 weeks, and also the front and back end implementation. Where as most of the group report task were left to be completed over a select number of days.<sup>[Gantt Chart]</sup>

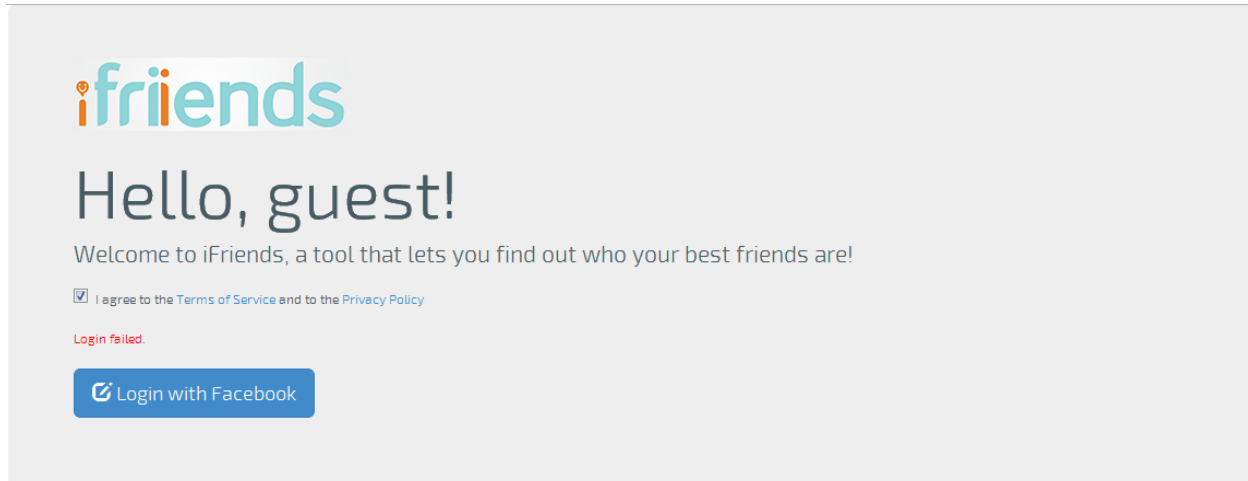
## Functional Components

### User End (Front)

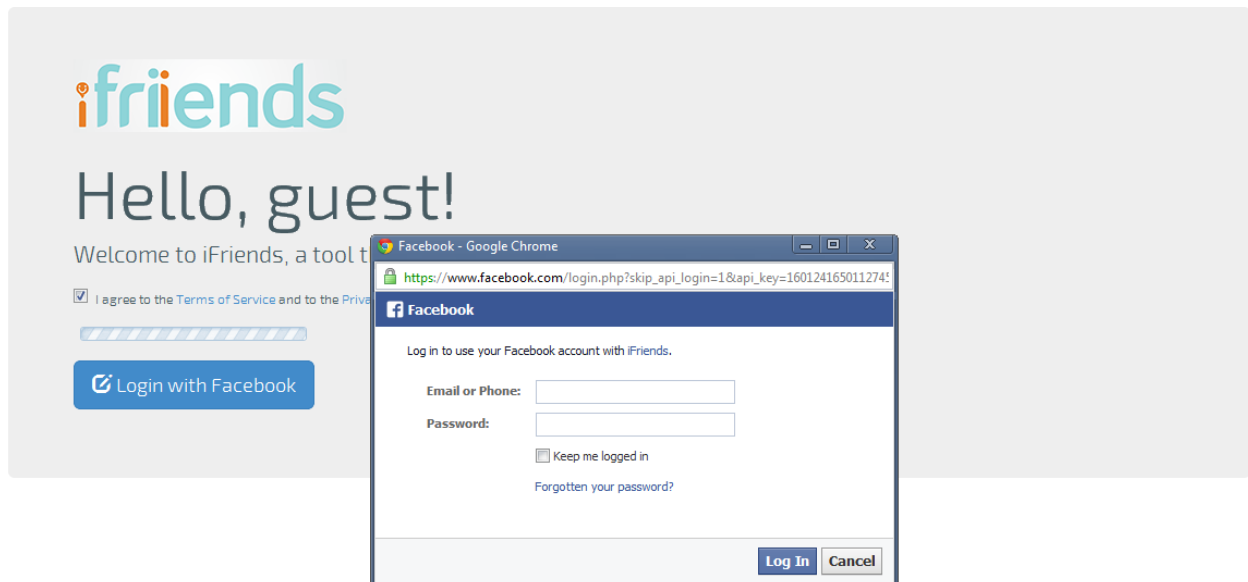
#### Login screen

On the login screen, the user is presented with a welcome message, a login button and a prompt to accept the Terms of Service and the Privacy Policy.

In order for the user to be able to initiate the login procedure, (s)he needs to accept those, by ticking the checkbox next to the statement “I agree to the Terms of Service and to the Privacy Policy”. If the user clicks the “Login with Facebook” button without ticking the checkbox, (s)he is presented with an error message, as agreeing to the Terms and Conditions and Privacy Policy is a legal requirement to use the software:

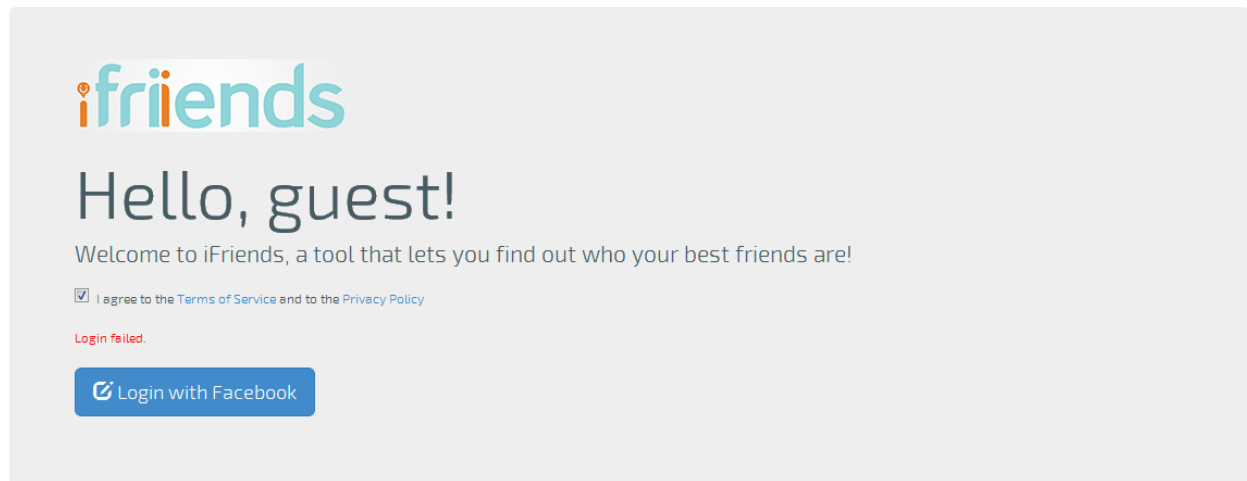


After the user accepts the Terms of Service and the Privacy Policy and clicks the “Login with Facebook” button, (s)he is presented with a popup window prompting for a valid Facebook username and password. A progress bar also appears in the place of the button in order to highlight that the login flow has started and it is in progress:



Should the user close the window or the login flow is interrupted for any other reason, a “Login failed” error message appears:



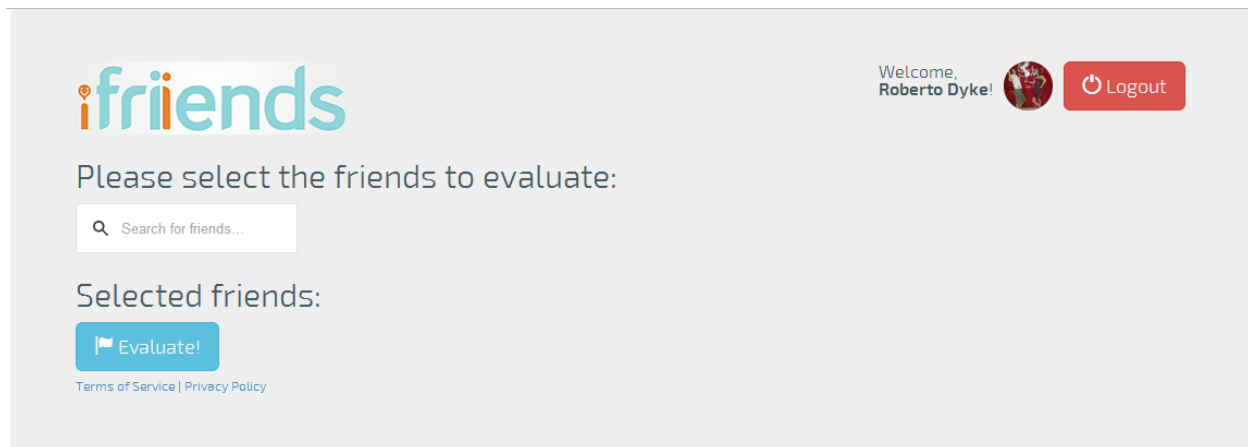


If the user enters a wrong username and / or password, another error message is displayed in the popup Window. The validation is done through the Facebook API.

The screenshot shows a Facebook login error popup. The header is a dark blue bar with the Facebook logo and the word 'Facebook'. Below the header, it says 'Log in to use your Facebook account with iFriends.'. The main content is a red-bordered box with the heading 'Please re-enter your password'. Inside the box, it says 'The password you entered is incorrect. Please try again (make sure your caps lock is off).' and 'Forgot your password? Request a new one.'. Below the red box, there are two input fields: 'Email or Phone:' with the value 'roberto.dyke' and 'Password:'. Below the password field is a checkbox labeled 'Keep me logged in' and a link 'Forgotten your password?'. At the bottom right of the popup are two buttons: 'Log In' and 'Cancel'.

### Main screen

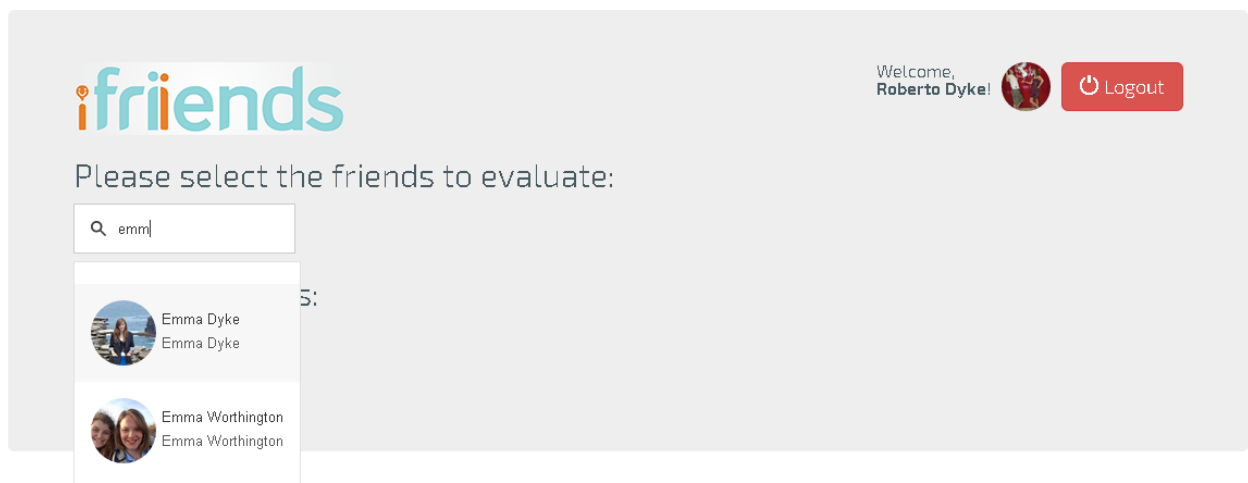
After the user successfully logs in, (s)he is presented with the main screen:



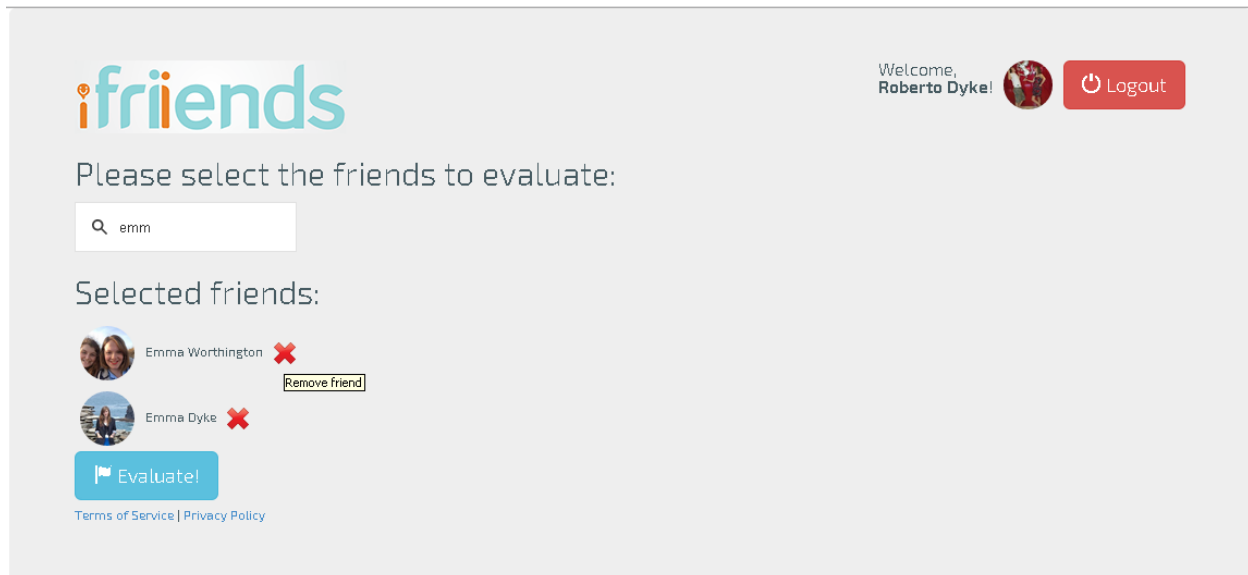
This presents the user with the following list of options:

- Search for friends, through the corresponding input text field;
- Evaluate the friends selected
- Logout, using the logout button on the upper-right side of the screen.

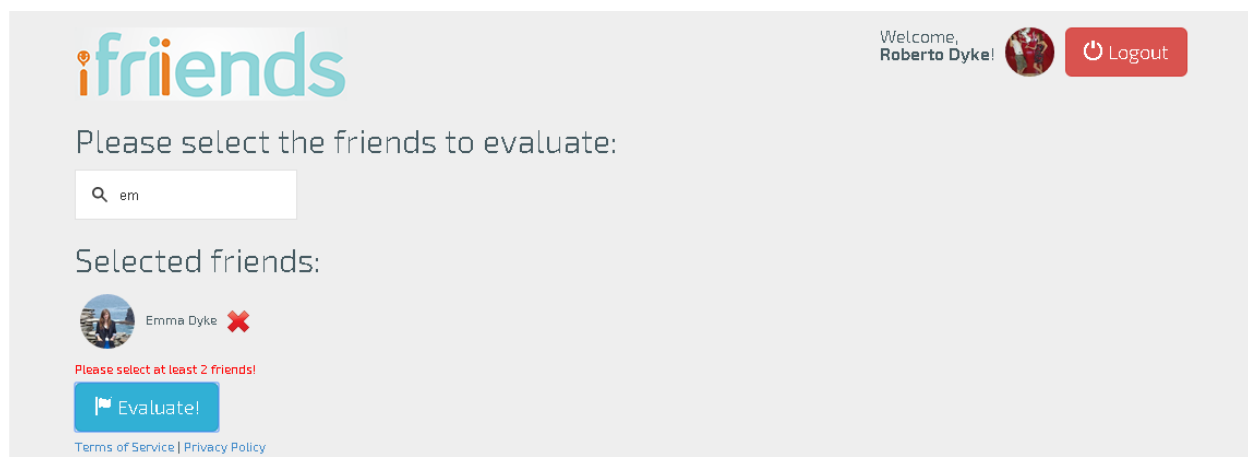
Using the input text search field, the user can input the first few letters of the friend names. The list of friends whose name match the text inserted is automatically displayed.



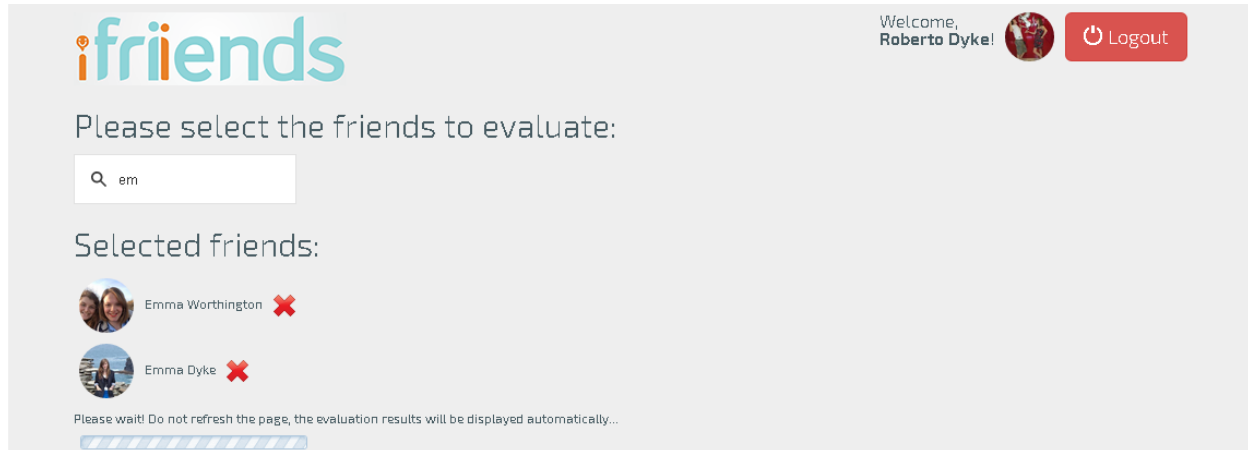
After the user clicks on the name of a friend, it is added to the list of friends for evaluation. The user can also remove one or more of the selected friends by clicking the “X” button next to each friends’ name.



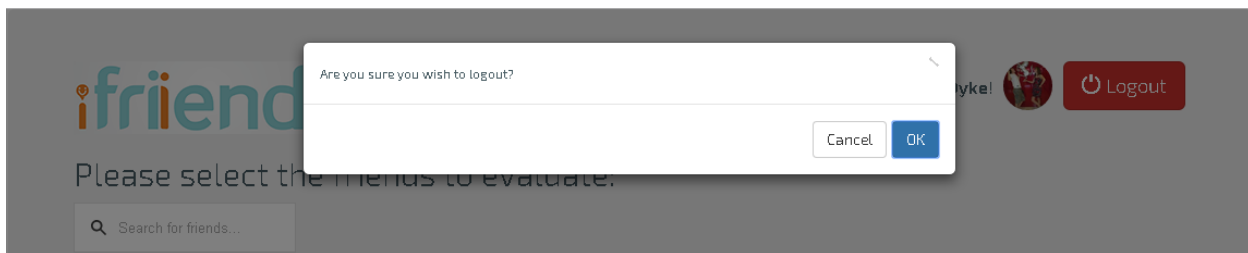
If there are not at least two friends selected (which is a requirement for the algorithm to work), an error message is displayed:



After clicking the "Evaluate" button, a progress bar is displayed, prompting the user not to refresh the page, as the results will be displayed automatically when the evaluation is finished:



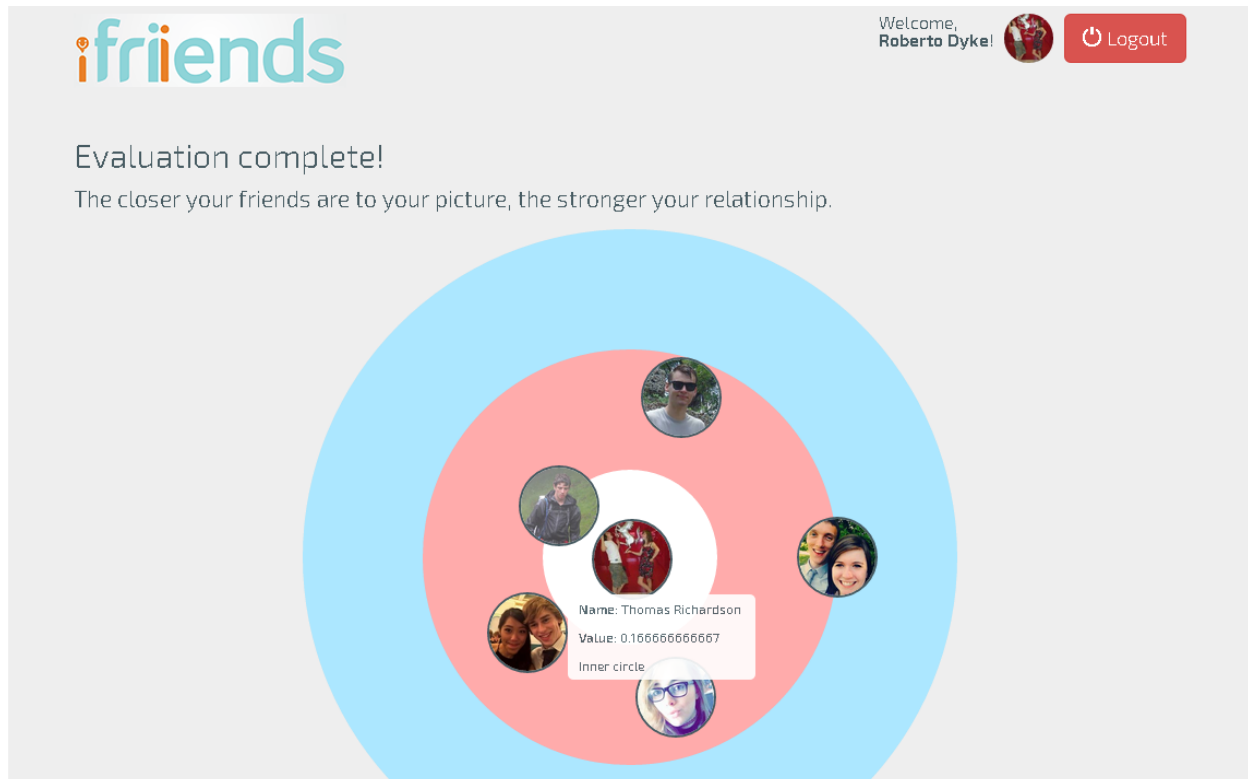
The user is able to logout of the software at any time, by clicking on the red “Logout” button, on the upper-right side of the screen. A confirmation message is displayed in the event the user clicked on the button by accident.



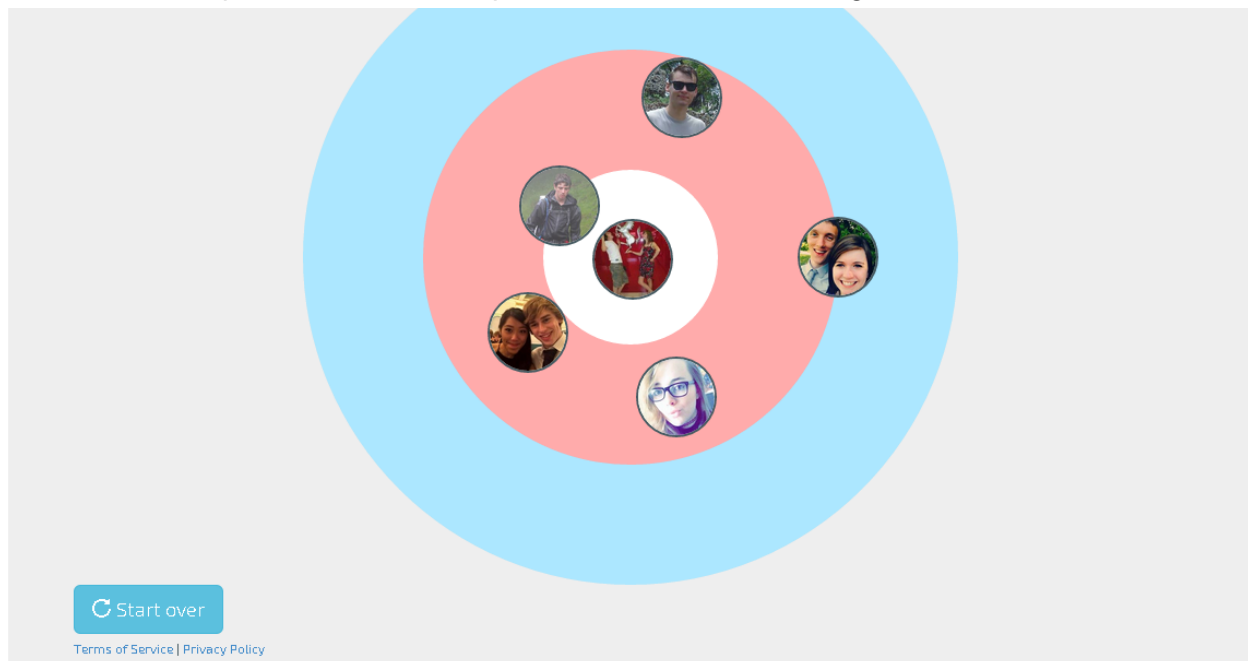
### Results screen

After the evaluation is completed, the user is presented with the evaluation results screen. A confirmation message is displayed highlighting the fact that the evaluation has completed. A short sentence explaining how to interpret the diagram is also included.

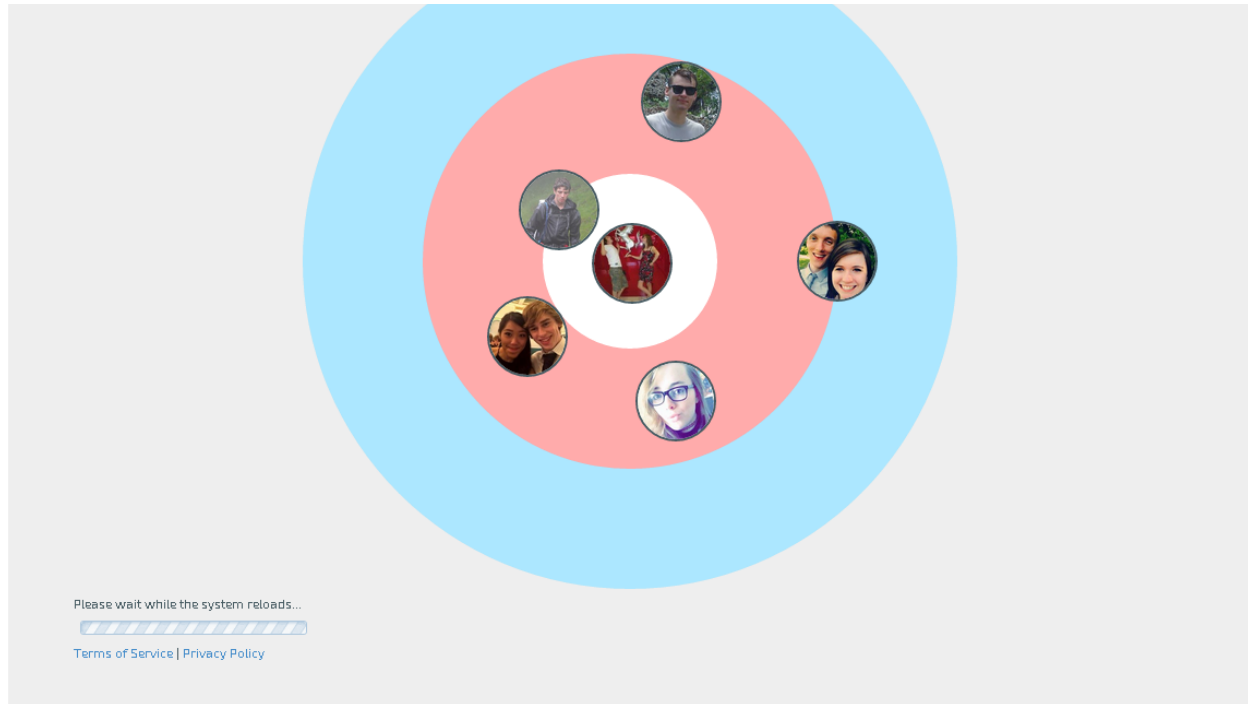
When the user hovers the picture of one of the friends, the following information is displayed in a tooltip: the name, the closeness value and whether the friend is in the inner circle or the outer circle of friends. When the user clicks on the profile picture of one of the friends, the Facebook profile opens of that person opens in new window / browser tab.



The user is also presented with the option to “Start over” and begin a new evaluation.



After the button is pressed, a progress bar is displayed that prompts the user to wait until the page reloads.



## Overall

The frontend is minimalistic, providing essential functionality of the software in a concise manner. It adheres both to the Facebook Brand Guidelines<sup>[7]</sup> and to the Human Computer Interaction [Usability Principles](#) listed in a subsequent section of the report.

## Developer End (Back)

### Application Authentication

There were many issues that had arisen when querying for Facebook data of a user's friend. Our software has not yet been authorised by Facebook meaning that it is difficult to access a friend's data that relates to personal information, private messages, photo albums, and certain other key pieces of information. This is mainly to protect Facebook users through the Facebook privacy policy for app makers that use the Facebook API. To access the friend's information they must first grant the app permission access to their personal information, the friend can simply do this by logging into the Facebook app. This would mean that we could test our system with real world data and get true results that reflect the chosen friendships. We also tackled an issue with the API which will be discussed in the "[Facebook Query Limits](#)" section of this report. The impact of these issues may be reduced by authenticating the app, which involves going through the authentication process, which requires vigorous app testing, a fully built and well documented app, and a detailed explanation of why a system requires the privacy settings it is requesting; we have not done this due to time constraints.

### Data Volumes

In the design report we failed to cover problems caused by the data volumes we would be handling, we have covered these issues here.

### Facebook Query Limits

By doing some additional investigation we have found that according to the latest version of the Facebook Platform Policy<sup>[1]</sup> 6.11. - “If you exceed 5M MAU, 100M API calls per day, or 50M impressions per day, you may be subject to additional terms.” We interpret this as just under 69500 call per minute, which is more than enough for a single user, when assuming users have an approximately 200 friends<sup>[2]</sup>. The program shall make a maximum of 90 calls per friend to the Facebook API, so on average there should be 1800 calls per user; thus allowing 38 users to calculate their friendship every minute.

### Server Requirements

To test the RAM requirements of the iFriends program we compared the server resource requirements w/o the program running to the server with a single instance of our program running.

		USAGE																	
		CPU			vMEM(MB)			pMEM(MB)			EP			nPROC			IO(KB/s)		
From	To	a	m	i	a	m	i	a	m	i	a	m	i	a	m	i	a	m	i
04-26 07:00	04-26 08:00	0	0	100	0.02	18.88	4096	0.05	8.14	1024	0	1	20	0	1	200	0	168	1024
04-26 08:00	04-26 08:05	0	0	100	7.43	17.84	4096	3.65	7.34	1024	0	1	20	0	1	200	0	4	1024
04-26 08:04	04-26 08:05	0	0	100	17.84	17.84	4096	7.33	7.34	1024	1	1	20	1	1	200	0	0	1024
04-26 08:05	04-26 08:06	0	0	100	0.00	0.00	4096	1.04	1.08	1024	0	0	20	0	0	200	0	0	1024
04-26 08:06	04-26 08:07	0	0	100	17.84	17.84	4096	7.30	7.33	1024	1	1	20	1	1	200	0	0	1024

To help synthesise this diagram I shall explain that the each row represents a snapshot of a given time period, the two we are interested in are the ones marked by the blue and green dot (the others are previous tests). The blue dot's row represents the server's peaks w/o the program running, while the green represents the server's peaks with the calculation running. The important columns are the “vMEM” and “pMEM”, these represent the amount of virtual and physical memory (in MB )the calculation is using. According to the diagram our program requires an additional 17.84MB virtual memory and 7.30MB actual physical memory. If we ignore the Entry Process limit (EP - concurrent processes) we can extrapolate that on this server we are limited to 4GB vMEM and 1GB pMEM, this would mean the theoretical maximum number of requests this server could handle is ~140 users, this constraint is caused by the limited amount of physical memory available.

### pThreads Test Server

In order to alleviate some of the pressure that our server would be under in a high-load scenario, we toyed with the idea of modifying PHP to allow us to utilise Posix Threads. This would have allowed us to potentially increase the speed of operations by taking advantage of

multiple threads, rather than just one. We achieved this with the PHP extension pThreads<sup>[5]</sup>. In order for us to take advantage of pThreads however, we first had to recompile PHP with support for Zend Thread Safety.

As we required a specialised build of PHP for the pthreads extension to work, we attempted to contact our current hosting provider to request they modify the PHP version used.

Unfortunately due to the performance constraints of shared hosting they declined our request (shared hosting doesn't usually contain many cores for multithreading). Our only remaining option was to host it ourselves.

Fortunately, one of our group members already had the hardware and bandwidth required to host our site as well as all the requisite software. Due to the nature of package installations using aptitude however, we were unable to overwrite/upgrade the current systems version of PHP, and therefore had to completely reinstall it. This proved to be a game of trial and error due to the many possible configuration options and settings given by PHP at build-time, and the lack of OS specific instructions for CentOS or RHEL.

After the arduous task of rebuilding and reinstalling PHP, we ran multiple tests with pthreads to ensure for promised speed and stability, and at one stage had over 500 threads successfully execute! Unfortunately we soon encountered issues with pThreads inability to properly close/join threads leading to massive memory leaks, which in turn would swell to fill the available memory and ultimately lead to a full system crash.

After extensive troubleshooting we found the underlying cause to pThreads issues, but decided that the benefit was greatly outweighed by the effort required to refactor all of our code and libraries to be thread-safe.

### System Compilation

As a team, we ran into several challenges when it came to producing this software, one of the biggest was writing code concurrently and getting it to easily fit together. Our group consisted of 7 programmers, so coordinating the distribution and compilation of code tasks was a big problem. By breaking the program down into functions, creating a specification for the inputs and expected output, along with a description of the process the function was expected to achieve, made this possible to achieve. The most significant code compilation was between the front and backend of the system, till this point the website's main page and subpages were all developed separately from the backend script, which simply took a list of user ids and returned the friendship strength of each user id. When we started writing the front end we treated the backend script as a massive function, this meant that when it came to put the two sections together the "oracles" of the sections came added in a small bit of POST code; this took a mere two hours to get fully working.



## Team Delegation & Structure

The team was organised in a decentralised (egoless) team structure, however as the weeks progressed the principles of the Mixed control team were also included in our team.

The reason for this was because the team felt that the principals of the approach were the ones that would most be beneficial to the team. These included the fact that decisions were shared - one thing that our group definitely adopted, as to resolve a problem, despite the fact that team had a group manager by the end of the project, as a team we worked together to overcome any problems faced.

Tasks were delegated within the team through the process of identifying different team member strengths and using these strengths to complete the tasks. Due to the team being a mixture of both Computer Science and Business Information Systems students, this then meant that some members felt more confident doing certain aspects of the project in comparison to others. To overcome this problem a meeting was held to discuss what members felt, and what should happen from now on, and it was agreed that report writing, research, meeting organisation, and other tasks would be delegated to BIS members, while other members of the team carried on with the implementation. Regular weekly meetings were held on a Friday where the agenda for the next week was discussed, and a brief outline for the tasks that would be uploaded on the Monday that would need to be completed. Every Monday a new set of tasks were uploaded to the shared drive for members to work on throughout the week until the next meeting.

One principle of a De-centralised egoless approach is that there is no project manager, and that this should be adopted by different people each week. This at the beginning seemed like a good idea, however as the project progressed, and grew larger, we as a team decided that a Project Manager would be more beneficial. We did help the Project Manager and managed own sections of the project delegated efficiently.

As with the design section, we used a Facebook group to coordinate work and discuss problems - along with the weekly team meetings - we had many routes to communicate about the project. We have used Gantt Chart software to illustrate our progress through the project - screenshots of the Gantt Chart may be found above in the Progression section. In the appendix, at the end of the report, you shall find a list of timings for different types of task, specifying the individual and the task. [\[timings\]](#)

## Modular Code Design & Its Impact On Group Work

The group have decided that the software should run with modularity, as it will enhance efficiency and run times. Since we have many programmers that prefer to be on the programming side of the software development we have human resources to do this. The modular design helps naturally split the code up into different segments. This means more resources can be used and each member can be assigned a certain module to complete.

This will provide many advantages to the software development process and will allow group members to individually upgrade their work without affecting the code that has been created by other users. The group are always working at a constant pace to finish work, and by introducing modularity, it will mean that the finished program can be tuned quickly and efficiently making sure everyone has mastered the specific code of their area in the program.

This code's initial modular design helped when assigning tasks, outlines of individual functions were written, along with PHP testbeds that allowed team members to test their code worked as expected. The test bed consisted of the function an individual was concerned with, and a mock version of the Facebook API class we planned on implementing. In the image below you can see the FB class has little to no actual functionality, and the data retrieved by it is hardcode, \$data\_1 and \$data\_2 are valid samples of the JSON strings returned by the Facebook API for the requests simulated.

```
1  <?php
2
3  class FB {
4      private static $data_1 = '{"data": [{"id": "123", "created_time": "2014-11-16T10:49:10+0000"}]}';
5      private static $data_2 = '{"data": [{"id": "789", "created_time": "2014-11-16T10:49:10+0000"}]}';
6
7      public static function getLinks($user_id, $params) {
8          if ($user_id == 123) {
9              return self::$data_1;
10         } else if ($user_id == 789) {
11             return self::$data_2;
12         }
13     }
14 }
15 // Change this function here
16 function functionName($user_id_1, $user_id_2) {
17     $json = FB::getLinks($user_id_1, "params"); // Gets data from FB API
18     $data_1 = json_decode($json, true); // Converts JSON form string into a php array
19     var_dump($data_1); // Prints php array (For testing purposes)
20     echo "<br/>";
21     $json = FB::getLinks($user_id_2, "params"); // Gets data from FB API
22     $data_2 = json_decode($json, true); // Converts JSON form string into a php array
23     var_dump($data_2); // Prints php array (For testing purposes)
24     echo "<br/>";
25
26     // Insert calculation here.
27
28     $result = 2;
29     return '{"functionName":' . $result . '}'; // Returns result in JSON form
30 }
31
32 functionName("123", "789");
33
34 ?>
```

Once the programmer had finished the function they added the script to a submissions folder, where the functions were thoroughly tested to check they worked as expected, code that passed this testing was added to a completed folder. This was done on a week-by-week basis.

Additional documentation was made for the front and back end components, we shall include all the "Programming Outline" documents and the related code in the zip file.

## Client Meetings

Client meetings were held mostly back in the Autumn semester. This was to ensure that the requirements that were formulated by the group met the requirements specified by the client. Meetings with the client were also been conducted during the spring semester, but less frequently, mainly to discuss the report, and to ensure the implementation of the work was on track, and that all members were contributing to the work.

## Overall Software System

### Changes to Design

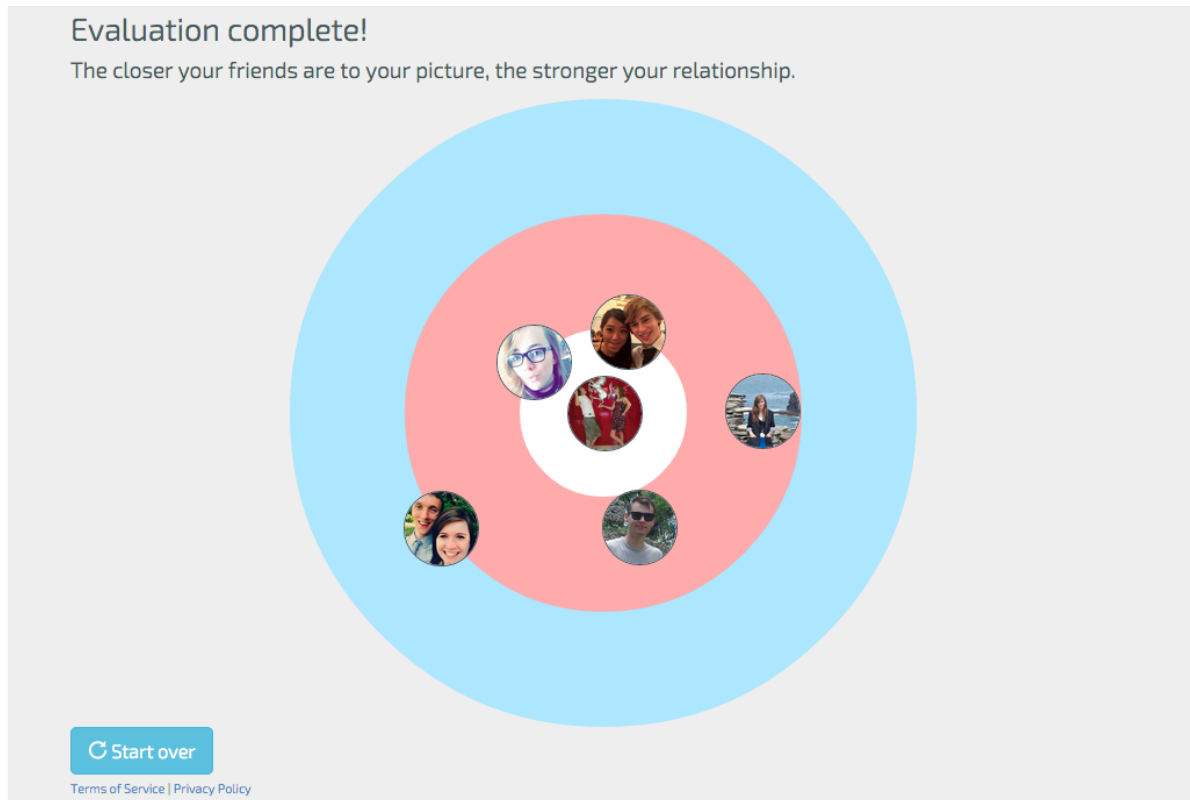
Since the initial design report, we have made some changes to the system architecture. Some of these changes have been improvements to improve the quality of the code, by creating objects with low coupling and high cohesion. Though some changes to the design have been due to the time constraints of the project.

We have produced a class diagram of the new class structure. [\[class diagram\]](#) The main alteration to the classes is the method scope, many of the internal methods have been made private with a single public method “run” that uses the internal methods, then returns the result. This made a lot of sense for the Algorithm class and friendship variables (“Functions”) class, the slightly altered approach provided an appropriate level of abstraction to the program, making the script especially easy to follow when debugging.

Due to time constraints the “FB” class was left incomplete, this meant that certain functions were missing, such as getPhotos, getStatus, and other features such as the ability to page Facebook data has not been completed. The inability to page Facebook data made the admin class redundant, so the front-end interface for the admin section has not been implemented, though the back-end code to control the admin section has been completed. This also had a knock-on effect on the “Functions” class, where **all** of friendship variables have been written and tested, but not implemented.

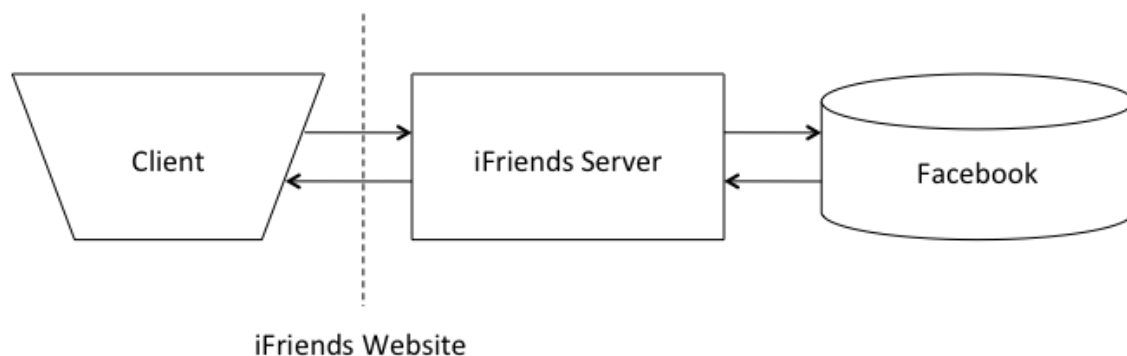
### AJAX

From a limited technical perspective we have implemented a combination of AJAX and hidden php scripts to produce the system’s user experience, which has been fine tuned in through a combination of group meetings and user feedback sessions.



AJAX has allowed us to produce a clean webpage that makes the system appear to be a single window, rather than a series of web pages. The client makes the initial request to calculate the friendship value through their browser, this calls a PHP script which runs the calculation, which passes the results onto a script to draw a diagram (illustrated above) of the results. The combination of HTML, CSS, JS and JQuery code to draw the diagram is then sent to the user through the AJAX tunnel, producing the final result.

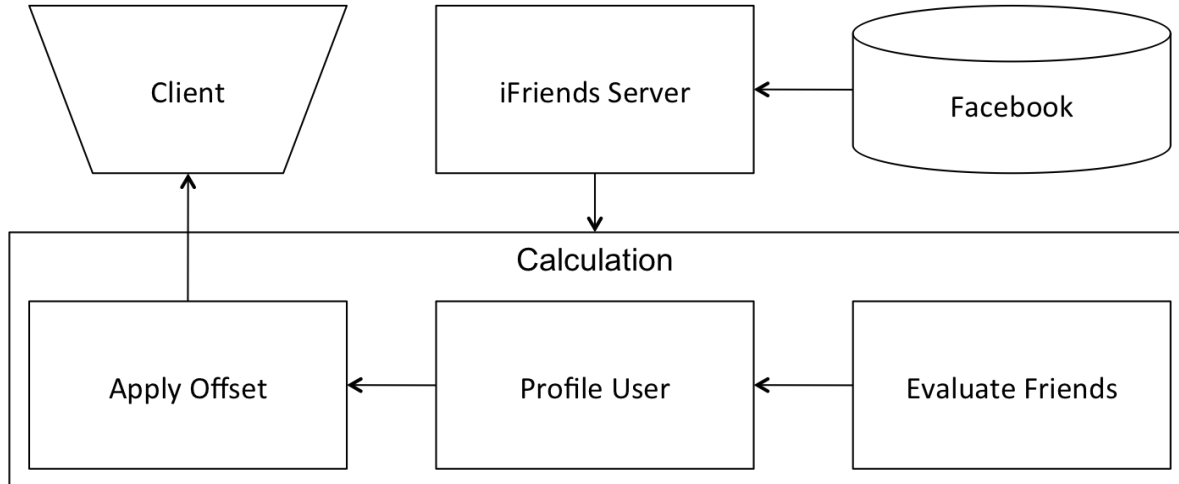
## Networking



Our product is interacted with by the user via the ifriends website. In order to use the features of the website, the user has to securely login to their facebook account. Although not

portrayed in the diagram above, the iFriends server is bypassed in order to do this. Once successfully logged in, the iFriends server will directly communicate with Facebook.

### Calculation Process



The diagram above gives an abstract view of the 3 stages of the algorithm that produce the list of friendship values.

The calculation that determines the strength of relationships comprises of three steps:

1. Profiling the user
2. Evaluating individual user-selected friendships, and then subtracting the average case from the user profile.
3. Finally applying an offset to the individual variables that make up the 'friendship value'

The final result is then used to determine the distance that the friends will be positioned from the user on the diagram.

### Variables

The following table of variables are from the design document, the variables of the table were collected from a research paper<sup>[3]</sup> found in the analysis stage of the project. Each of the variables named below has been converted into an individual function that our system can use to produce a numerical value. Documentation on each of these variables is available upon request, and has been removed from the document to reduce tediousness.

#	Name	Factor affected
1	Wall words exchanged	Intensity
2	Participant-initiated wall posts	Intensity

3	Friend-initiated wall posts	Intensity
4	Inbox messages exchanged	Intensity
5	Inbox thread depth (number of messages sent between pair)	Intensity
6	Participant's status updates	Intensity
7	Friend's status updates	Intensity
8	Participant's photo comments	Intensity
9	Friend's photo comments	Intensity
10	Participant's number of friends	Intimacy
11	Friend's number of friends	Intimacy
12	Days since last communication	Intimacy
13	Wall intimacy words	Intimacy
14	Inbox intimacy words	Intimacy
15	Appearances together in photo	Intimacy
16	Participant's appearances in photo	Intimacy
17	Distance between hometowns (miles)	Intimacy
18	Distance between Time zones (hours)	Intimacy
19	Friend's relationship status	Intimacy
20	Days since first communication	Duration
21	Links exchanged by wall posts	Reciprocal Services
22	Applications in common	Reciprocal Services
23	Number of mutual friends	Structural
24	Groups in common	Structural

25	Norm. TF-IDF of interests and about	Structural
26	Wall & inbox positive emotion words	Emotional Support
27	Wall & inbox negative emotion words	Emotional Support
28	Age difference	Social Distance
29	Number of occupations difference	Social Distance
30	Educational difference (degrees)	Social Distance
31	Overlapping words in religion	Social Distance
32	Political difference (scale)	Social Distance

All the variables specified have been implemented as specified in earlier documentation - available, but not included in the design report document - except for no. 22, which had to be modified. Since the Facebook API did not provide an appropriate solution to finding the applications in common, we changed the variable to find the games two people had in common. The 'applications in common' variable was part of the reciprocal services factor, which made up 7.9% of the result. We predict that this small alteration has little effect on the final result, due to its low prominence in the final result.

#### Variable Contingency

When implementing the solution we found that some user information was unretrievable. Because of this, important calculations that were required in the specifications could not function. For a work around, we constructed a solution that allowed the calculation to ignore the missing values and compensate for the change.

#### Missing Functions

Many of the friend variable functions specified above were not implemented in the final version of the code, though all the functions have been written they have not been added to the code. To illustrate how simple it is to add an extra module we shall show you the process to add an extra function.

In this example we take the Educational Difference function and add it to the program.

Step 1. The function must initially be written

```

1  function overlappingWordsInReligion($user_id_1, $user_id_2) {
2      $data_1 = $this->fb->getProfile($user_id_1,"religion")["religion"];
3      if (!isset($data_1))
4          return null;
5      $data_2 = $this->fb->getProfile($user_id_2,"religion")["religion"];
6      if (!isset($data_2))
7          return null;
8
9      // User_1
10     // $get_data_user_1 stores the string contained inside religion
11     $get_data_user_1 = $data_1["religion"];
12
13     $user_1_words = array(); // Array to store the words from religion t
14
15     // Get words from user_1
16     $delim = ' ,.!?;:()'; // Get rid of punctuation - didnt use reg exp
17     $tok = strtok($get_data_user_1, $delim);
18
19     while ($tok !== false) {
20         $tok = strtolower($tok); // Lower the case
21         array_push($user_1_words,$tok); // Store the word in the next in

```

Step 2. The code must be added the Functions class

```

187     }
188 }
189
190 class Functions {
191     private function overlappingWordsInReligion($user_id_1, $user_id_2) {}
192     public function __construct($token, $variables) {
193         $this->fb = new FB($token);
194         $this->variables = $variables;
195     }
196     public function run($user_id, $friend_id) {
197         $values = array();

```

Step 3. An if statement in the run method in the Functions class must be created

```

221     }
222     if (in_array("ageDifference",$this->variables)) {
223         $values["ageDifference"] = $this->getAgeDifference($user_id,$friend_id);
224     }
225     if (in_array("numberOfOccupationsDifference",$this->variables)) {
226         $values["numberOfOccupationsDifference"] = $this->numberOfOccupationsDifference($user_id,$friend_id);
227     }
228     if (in_array("overlappingWordsInReligion",$this->variables)) {
229         $values["overlappingWordsInReligion"] = $this->overlappingWordsInReligion($user_id,$friend_id);
230     }
231
232     return $values;
233 }
234 private function overlappingWordsInReligion($user_id_1, $user_id_2) {
235     $data_1 = $this->fb->getProfile($user_id_1,"religion")["religion"]; // Gets c
236     if (!isset($data_1))
237         return null;
238     $data_2 = $this->fb->getProfile($user_id_2,"religion")["religion"]; // Gets c
239     if (!isset($data_2))
240         return null;
241     ...

```



Step 4. The method name must be added to a list in the Algorithm class

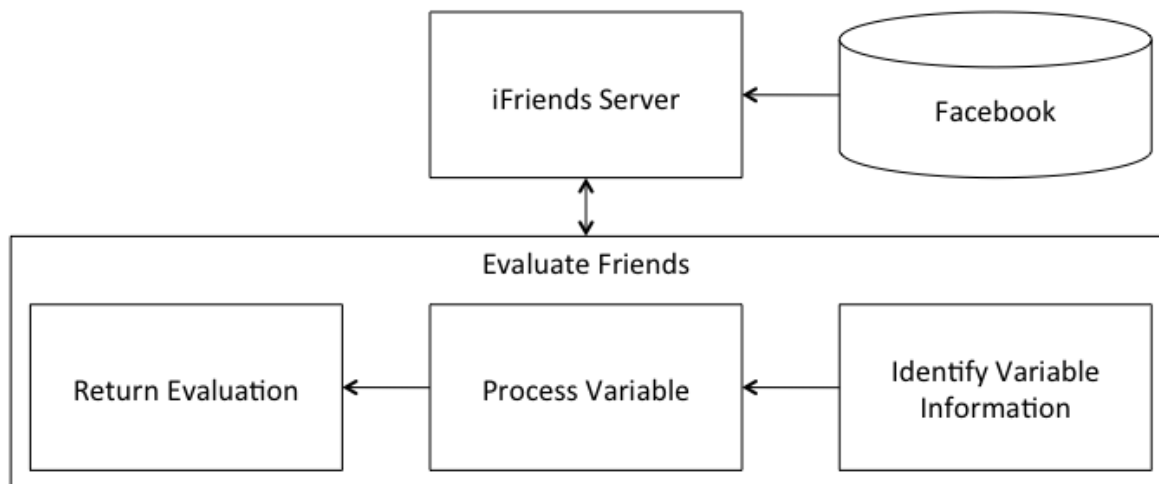
```

432
433 class Algorithm {
434     public function __construct($user_list,$token) {
435         $this->user_id = $user_list[0];
436         $this->friends_list = array_slice($user_list, 1, count($user_list)-1); // Extra
437         $this->friend_evals = new FriendEvals($this->friends_list);
438         $this->token = $token;
439         // // // // // // // //
440         // // // // // // // //
441         // Remove variables from this list for the algorithm to ignore them
442         $this->variables = ["daysSinceLastCommunication","daysSinceFirstCommunication",
443         "differenceBetweenTimezones","distanceBetweenHometowns","politicalDifference","numberOfMutualFriendsDifference",
444         "numberOfOccupationsDifference","overlappingWordsInReligion"]; // Variables used
445         // // // // // // // //
446         // // // // // // // //
447     }
448     public function run() {
449         if (count($this->friends_list) < 1) { // Verifys that friends list size is adequate
450             echo "Error - Not enough friends selected.";
451             return null;
452         }
453     }
454 }

```

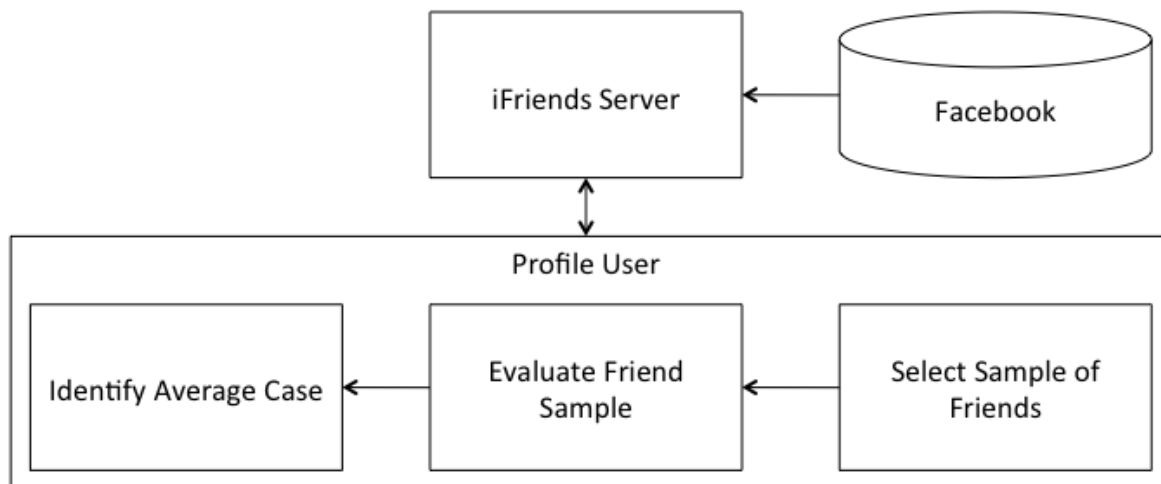
And done! The process for adding extra modules is very simple, this kind of design is very good for maintaining the code, especially when adding or debugging the friendship variables. This simplified approach has severely reduced potential headaches when slotting code together, though it is not without its intricacies - being you must add a string to the algorithm class - it is the code is relatively plug & play.

### Evaluate Friends



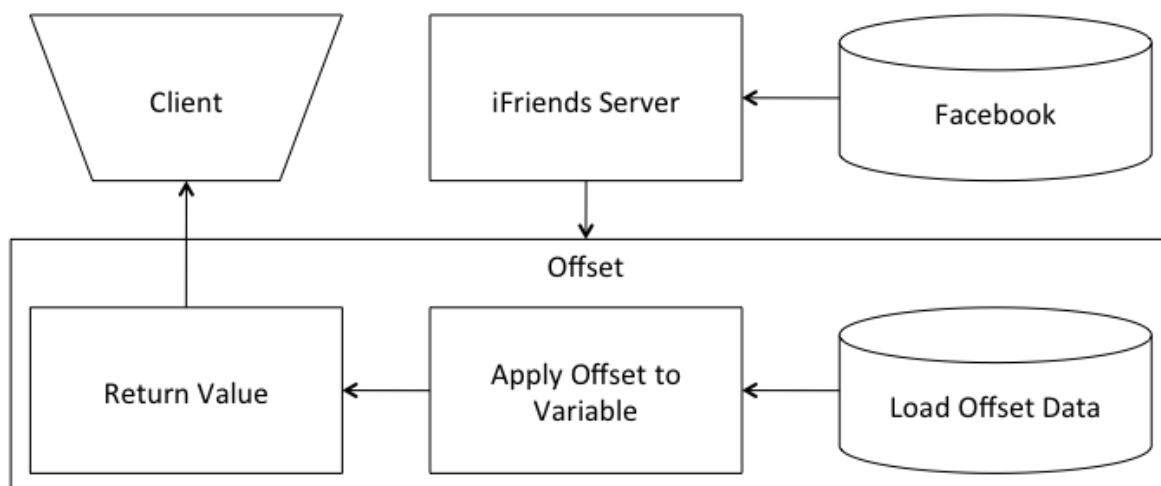
The Functions class gets information for each variable about a friend, and compares it - as specified by documentation about variables. On an abstract level the program attempts to get as much information about a user and their friends and calculates the results of the friendship variables from the available data. This is returned as an array of results for each variable analysed for each friend.

## Profile User



A profile of the user's habits is then made, this is done by taking the average value for each friendship variable across all the selected friends. Originally this function followed the diagram above, where it selected random friends to create this profile, we found that the performance impact wasn't significant enough for this sampling method to be necessary. This has reduced the variance present in our results.

## Offset



The offset value was a component specified in the design report, offsets dictate the significance of a friendship variable, offsets usually fell between -1 and 1 a value based on the properties of a variable. Since some friendship variables has an inverse effect on strength of friendship the offset value may be negative. Due to the offset value being a multiplier, the closer to 0 the offset value, the smaller impact the variable has on the final 'friendship value'. Once the offset is applied the sum of the product created from each variable is added, this is

done across all friends, once normalised we have retrieved our friendship value. This is returned from the algorithm as list of friends and friendship values, as illustrated above.

Modifying offset values can be done in the Algorithm class, each variable has an offset that either negatively affects the final result, or positively affects it. The offset value also indicates the significance of each value on the final result, timezone does not highly affect the final result, so in the image below you can see that the offset causes it to have a smaller impact on the final result.

```

communication")) {
  ation"] = log($scores["daysSinceLastCommunication"] + 1) * -1;

communication")) {
  ation"] = log($scores["daysSinceFirstCommunication"] + 1) * 1;

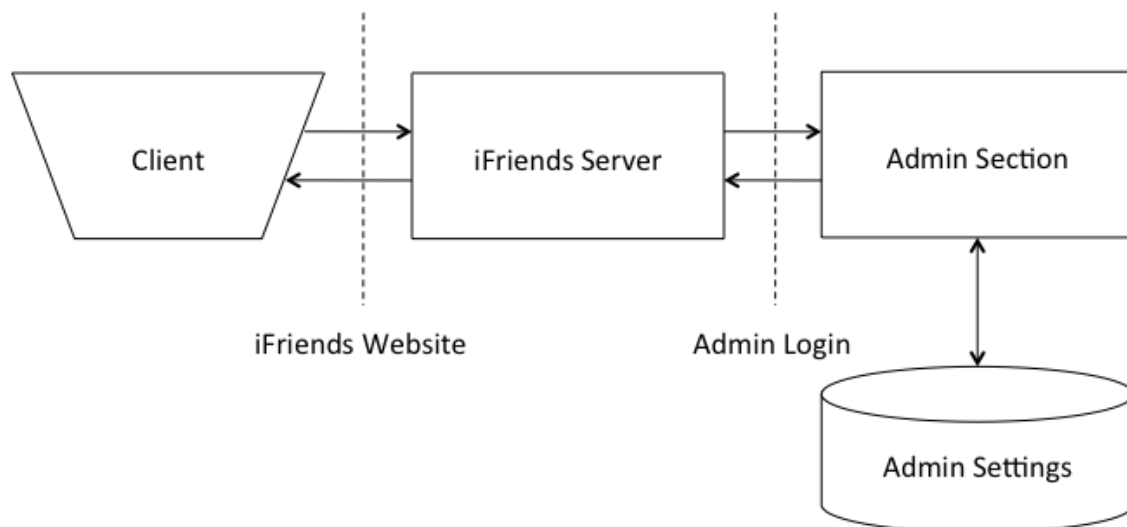
Timezones")) {
  zones"] = ($scores["differenceBetweenTimezones"] / 24) * -0.25;

Hometowns")) {
  vns"] = ($scores["distanceBetweenHometowns"] / 7926) * -7000; // 7926 is the equatorial

```

*(Note: Potential for integration with machine learning algorithm here, current program doesn't accommodate for training sets or feedback for learning.)*

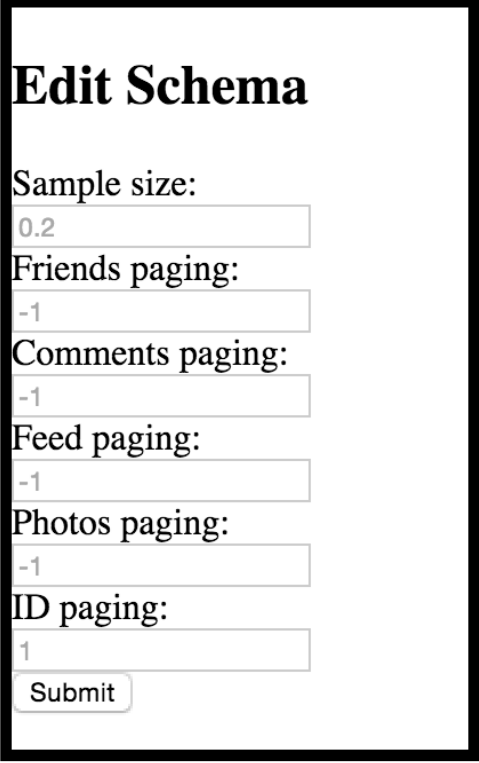
## Admin Section



We planned to provide an admin section where an administrator could control settings relating to the system. This would have involved a login page to authenticate the user is a genuine administrator, with a web page to behind that barrier to interface with a settings file - as

illustrated above. The admin was to be able to adjust amount of paging and the offsets for each variable.

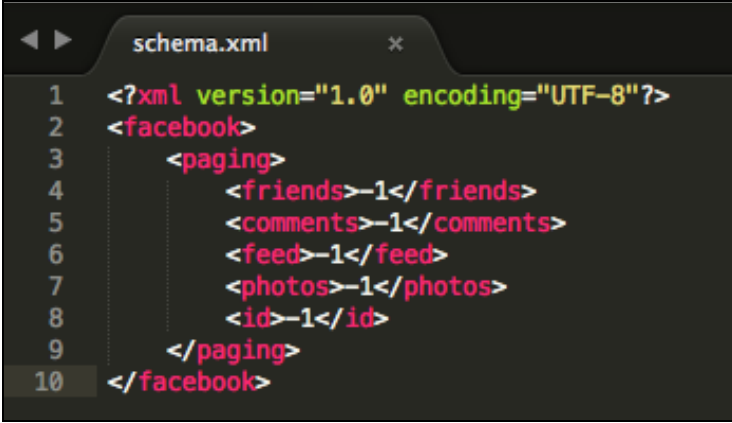
Unfortunately, the front end component of the settings file has not been implemented due to the “FB” class failing to use paging and no longer was required to specify the sample size. Below is a screenshot of an early prototype of the admin section, designed to interface with the schema.xml file. For this to be satisfactorily implemented the styling would still need to be changed, and a secure administrator login would also need to be developed.



The screenshot shows a web form titled "Edit Schema". It contains several input fields for configuration: "Sample size:" with a value of "0.2", "Friends paging:" with "-1", "Comments paging:" with "-1", "Feed paging:" with "-1", "Photos paging:" with "-1", and "ID paging:" with "1". A "Submit" button is located at the bottom of the form.

The code for the admin section shall be appended in the Zip file.

### Settings File



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <facebook>
3   <paging>
4     <friends>-1</friends>
5     <comments>-1</comments>
6     <feed>-1</feed>
7     <photos>-1</photos>
8     <id>-1</id>
9   </paging>
10 </facebook>
```

The settings file contains paging values for each type of Facebook data we are interested in. It can be edited whilst a user is calculating their result to tailor the results to the specific user preference. The change is effective to all users that click the 'calculate button' after the required change is chosen.

Originally the settings file contained a sample size value. This has since been removed as it is no longer used. A better method for retrieving information about users was found. This method requires no sample size and uses less facebook requests, in turn significantly reducing the calculation time.

## Testing

### Developers Perspective

The tester makes sure that all of the required features and functions are present and accounted for; this is done via testing from a developer's perspective.

For the purposes of testing, we created a set of test users in order to protect the real, sensitive data of Facebook users. The group of test users is encapsulated and does not interact with the rest of the Facebook social network. Relevant information for testing (ages, political affiliation, messages, etc.) used by the friendship evaluation algorithm was manually generated and associated with the testing users profiles, which represent genuine Facebook users in a controlled environment.

The credentials for the main test account is:

**username:** [roberto.dyke@gmail.com](mailto:roberto.dyke@gmail.com)

**password:** password

### Back End

In total 9 test cases were carried out for the development of the back end. 4 test cases were failed and 5 were passed.

Test cases that failed:

Test Case ID	Summary	Action
001	To test the API - Get all Inbox messages	Incomplete, add functionality to page data in the next version
002	To test the API - Get all Timeline (Wall) posts	Implemented method for the next version.
003	To test the API - Get all Status posts	Implemented method for the next version.
004	To test the API - Get all Photo information	Implemented method for the next version.

Test cases that passed:

Test Case ID	Summary
005	To test the API - Get all Friend information
006	To test the API - Get all User Profile information
007	To test the information acquired through the API is correctly converted to a computable form that can be implemented with our Friendship Algorithm.
008	User profile values are dependent on the specific friends analysed in application, meaning the profile has a greater degree of accuracy when more friends are added.
009	Presentation of results accurately reflects the results calculated by the algorithm

Test Case ID	#001
Test Case Summary	To test the API - Get all Inbox messages
Prerequisites	User is logged in, we have permission from user to access Inbox messages
Test Procedure	1.Select the user(me)/friend 2.Systematically request messages 3.Iteratively receive all messages
Test Data	TestUser account
Expected Result	All of the user(me)/friend's messages are retrieved

Actual Result	Gets the first page of the inbox messages
Status	Test failed
Remarks	Incomplete, must add functionality to page data.
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Test Case ID	#002
Test Case Summary	To test the API - Get all Timeline (Wall) posts
Prerequisites	User is logged in, we have permission from user to access Timeline posts
Test Procedure	1.Select the user(me)/friend 2.Systematically request Timeline posts 3.Iteratively receive all Timeline posts
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Timeline posts are retrieved
Actual Result	Program does not get Wall posts

Status	Failed
Remarks	Not implemented, method not yet written.
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Test Case ID	#003
Test Case Summary	To test the API - Get all Status posts
Prerequisites	User is logged in, we have permission from user to access Status posts
Test Procedure	1.Select the user(me)/friend 2.Systematically request Status posts 3.Iteratively receive all Status posts
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Status posts are retrieved
Actual Result	Does not get user status posts
Status	Failed



Remarks	Not implemented, method not yet written.
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Test Case ID	#004
Test Case Summary	To test the API - Get all Photo information
Prerequisites	User is logged in, we have permission from user to access Photo information
Test Procedure	1.Select the user(me)/friend 2.Systematically request Photo information 3.Iteratively receive all Photo information
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Photo information are retrieved
Actual Result	Does not get photo information
Status	Failed
Remarks	Not implemented, method not yet written.

Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Zain Tahir
Date of Execution	29/04/2015

Test Case ID	#005
Test Case Summary	To test the API - Get all Friend information
Prerequisites	User is logged in, we have permission from user to access Friend information
Test Procedure	1.Select the user(me)/friend 2.Systematically request Friend information 3.Iteratively receive all Friend information
Test Data	TestUser account
Expected Result	All of the user(me)/friend's Friend information are retrieved
Actual Result	Method gets friend's information from Facebook
Status	Passed
Remarks	Method works 100%
Created By	Zain Tahir, and Roberto Dyke

Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Test Case ID	#006
Test Case Summary	To test the API - Get all User Profile information
Prerequisites	User is logged in, we have permission from user to access User Profile information
Test Procedure	1.Select the user(me)/friend 2.Systematically request User Profile information 3.Iteratively receive all User Profile information
Test Data	TestUser account
Expected Result	All of the user(me)/friend's User Profile information are retrieved
Actual Result	Method gets profile information from Facebook
Status	Passed
Remarks	Method works 100%
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015

Executed By	Roberto Dyke
Date of Execution	28/04/2015

API-to-Algorithm Conversion

Test Case ID	#007
Test Case Summary	To test the information acquired through the API is correctly converted to a computable form that can be implemented with our Friendship Algorithm.
Prerequisites	We have successfully acquired the correct information through the social media site's API.
Test Procedure	1.Input test data 2.Process test data 3.Output test data
Test Data	See Friendship Algorithm design for specification of data set to pass through
Expected Result	Computable result for friendship variable functions
Actual Result	Result is computable
Status	Passed
Remarks	FB API methods implemented all return data in form expected
Created By	Zain Tahir, and Roberto Dyke

Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Algorithm

Test Case ID	#008
Test Case Summary	Ensure the generated profile of the user is accurate
Prerequisites	We have successfully acquired the correct information through the social media site's API.
Test Procedure	1.Input test data 2.Process test data 3.Output test data
Test Data	See Friendship Algorithm design for specification of data set to pass through.
Expected Result	Result should follow the mathematical equation specified (in the Friendship Algorithm design specification).
Actual Result	Mathematically equation followed
Status	Passed
Remarks	User profile values are dependent on the specific friends analysed in application, meaning the profile has a greater degree of accuracy when more friends are added.

Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

Display Information

Test Case ID	#009
Test Case Summary	Ensure the diagram of the data is accurate to the result calculated by the algorithm
Prerequisites	We have successfully run the algorithm and have accurate results to display
Test Procedure	1.Input test results 2.Draw test results on canvas 3.Display canvas on the user's screen
Test Data	See Friendship Algorithm design for specification of data set to pass through.
Expected Result	Result should accurately depict the results from the Friendship Algorithm
Actual Result	Presentation of results accurately reflects the results calculated by the algorithm
Status	Passed

Remarks	Diagram works 100%
Created By	Zain Tahir, and Roberto Dyke
Date of Creation	04/02/2015
Executed By	Roberto Dyke
Date of Execution	28/04/2015

### Functions

Each week some members of the team were delegated functions to develop that took information from Facebook's graph API and that data was manipulated to the specified requirement of that particular function. These functions included the number of inbox messages exchanged between two users, how many mutual friends two user had etc. Smoke Testing was carried whenever a function was completed and received for testing. To test that these functions worked correctly and return the correct data, test data was developed by Roberto Dyke for each function by using the Graph API Explorer to pull JSON data from the API which was inputted into the function and the output was checked to see if it was returning the correct result.

The alternative to this method of testing of the function would have been to create data ourselves and test the function using that but by using the API to generate test data from actual an user's data, it insured that the function would work without any problems when integrated into the system, which would use the API to pull data from Facebook.

### User's Perspective

User testing is valuable to determine whether the application will be popular and successful. The testers were chosen on the basis on the fact that they have had no contribution towards the project, this is done to ensure that there is no familiarity comes into play when the tester are carrying out the test cases and thus not allowing them to perform the task with more ease than someone who hasn't never heard about the system.

The think aloud method was used to test from the user's perspective because of how convincing it is. Direct exposure to how users think about the work done by the front end team. Getting the team to sit in on a few thinking-aloud sessions doesn't take a lot of their time and it was the best way to motivate them to pay attention to usability.

To test the front end, two think aloud tests were carried out.

Think aloud website review
User ID 001
<p>Whole system: No description of application's purpose - lacking context.</p> <p>Login screen: Took some time to log in.</p> <ul style="list-style-type: none"> <li>- Lacking professional finish</li> <li>- Lacking identity</li> <li>- Missing links to an about, t&amp;cs, privacy policy page</li> </ul> <p>Options screen: Would like some verbal validation that login was successful</p> <ul style="list-style-type: none"> <li>- Doesn't understand what remove user button mean; maybe change the sprite image</li> <li>- Unappealing spacing between the user's image and their name in the list of selected friends</li> </ul> <p>Results screen: Either put the results on a new screen or get the page to scroll down</p>
User ID 002
<p>Whole system: No branding</p> <ul style="list-style-type: none"> <li>- Not particularly appealing</li> <li>- There's no heart</li> </ul> <p>Login screen: fix spelling errors, provide more context</p> <p>Options screen: create a longer search bar</p> <ul style="list-style-type: none"> <li>- Remove search results, if there are no results</li> <li>- No keyboard navigation</li> <li>- Do not like the appearance of the search box</li> <li>- Hover colour must be darker - hard to see small change</li> <li>- Logout button should be in the top right to conform with external consistencies</li> <li>- Login button does not adhere to facebook api terms</li> <li>- Again doesn't understand what remove user button mean; maybe change the sprite image</li> </ul> <p>Logout: should provide verification dialogue</p> <ul style="list-style-type: none"> <li>- Should have alert for when the user logs out successfully</li> <li>- Received error code on logout screen "Warning: Cannot modify header information - headers already sent by (output started at /home/ifriwqgi/public_html/frontend/index.php:61) in /home/ifriwqgi/public_html/frontend/index.php on line 140" - Didn't successfully logout, was still logged in</li> </ul>

Taking into account these two think aloud tests changes were made to the front end to make it more user friendly. Both users complained that the application lacked a sense of identity, this was fixed by including a logo and there were further suggestion on how to improve the design such as increase spacing in certain areas, changing the colouring scheme and maintaining design consistencies, these were also considered in order to improve the design of the front end.



## Test Cases

We decided that a simple table format would best suit the client side testing as it will be easy for the user to follow and enter the results. The results are anonymised to comply with laws and regulations. Each user will be given a unique user ID to allow results to be located efficiently. These were carried out after the think aloud tests, so some design changes had implemented.

<b>LOGIN SCREEN</b>					
<b>User ID</b>	001				
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for Expected vs. Actual (10 being Best)</b>	<b>Additional Comments</b>	<b>Time Taken</b>
Login Screen	Provide login button to Facebook	Provide login button to Facebook	7	Login screen lacks content. Perhaps provide a brief description about it.	
Details required	Facebook credentials (username & password)	Facebook credentials	10	Standard so it's good.	
Feedback throughout process			10	I like the load bar, i.e. good indication.	
Time taken to Complete Login					

<b>Failed To Login Screen</b>					
<b>User ID</b>	001				
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for</b>	<b>Additional Comments</b>	<b>Time Taken</b>

			<b>Expected vs. Actual (10 being Best)</b>		
Feedback on Login error	Warn user that they have failed to log in	Warns user, they have failed to log in	10	Sufficient indicator that my login failed.	
Re-enter of details	Allow user to attempt to log in again	Allows user to attempt to log in again	10	Good!	
Feedback once successful			10	Very good acknowledgement that I successfully logged on.	
Time Taken to login after error					

<b>Permissions Screen</b>				
<b>User ID</b>	001			
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for Expected vs. Actual (10 being Best)</b>	<b>Additional Comments</b>
Permissions Asked	Ask user for permissions	Permissions asked of user	9	Yeah I work bud.

<b>Options Screen</b>				
<b>User ID</b>	001			
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for Expected vs. Actual</b>	<b>Additional Comments</b>

			<b>(10 being Best)</b>	
Options Available	Allow user to select friends to evaluate	Allows user to select friends to evaluate	8	I like it! Very nicely presented.
Options available to change presentation	Allow user to change the presentation of the results	User can hover over node to view more information, but nothing more	8	It looks like the option screen blocks the evaluation button so I need to click out first to evaluate. Additionally, the minus sign to remove a selected friend is difficult to determine.

<b>Results Screen</b>					
<b>User ID</b>	001				
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for Expected vs. Actual (10 being Best)</b>	<b>Additional Comments</b>	<b>Time Taken</b>
Expected Layout	The user's results are more-or-less what they expected	The results are displayed the way the user expected	6	Minimalistic structure to the website.	
Results Displayed readable?	User can interpret results	Results are clear to user	8	Yes.	
Accuracy of Results	Results correctly represent user's friendship	Close enough	8	It's alright like.	
Processing time to calculate results					

<b>Post Results Screen</b>				
<b>User ID</b>	001			
<b>Subject</b>	<b>Expected</b>	<b>Actual</b>	<b>Mark / 10 for Expected vs. Actual (10 being Best)</b>	<b>Additional Comments</b>
Options to Share results	Provide link to publish results on Facebook	Not implemented	0	Idiot.
Navigation available after results	Provide navigation	Provides user navigation to restart or logout	6	Need to scroll down to start over, i.e. do another evaluation.
Overall Satisfaction			8	Functionality wise, the website is excellent, however, it felt like the website requires to be faster when evaluating and needs more content.

#### Summary of Test Case (User 001)

**Login Screen:** The tester commented on how the Login screen lacks content. To improve in the next version a brief description of the app will be added to the login page.

**Failed To Login Screen:** The failed to login section works as expected. Upon entering incorrect details the user is given an error and asked to input the correct details.

**Results Screen:** The user commented that the results screen was too minimalistic, this can be rectified by adding a brief description on how the results are calculated in the next version of the system.

**Post Results Screen:** To see the actual results the user has to scroll down the page which the user didn't like as they wanted it to be appear right in front of their eyes. Also there was no option to share the results, this needs to implement in the next version of the system.

#### Quality of code

To produce good quality of code, the implementation team had two main rules to adhere to:

1. It should be legible - The code should clearly state the intent. If the reader can't make sense of the code. This was met certainly for the individual functions that were developed as the code had to be tested by a team member who had no input in their development.

2. It should be testable - The code should be organised in a way that facilitates unit testing. Each function had arguments. The test data was inputted and the outcome was compared to the expected outcome deduced from the test data.

## Justification/Evaluation

### Justification for design choices

While designing the user interface, we had to consider the different principles that would affect the usability of the application, namely: state visibility, feedback, constraints, application mapping, experience consistency, affordance, system predictability, synthesisability, and familiarity.

#### Visibility

All the actions available to the user are visible and obvious. On the login screen, the user is presented with the option to Login, after accepting the Terms of Service and Privacy Policy. The login button is clearly visible, being clearly labeled with a visible font size. The same applies to all the buttons and options across the software. On the main screen, the user is presented with the option to choose the friends to evaluate and manage the list, by adding and removing friends. The logout option is also presented in the upper-right side of the screen.

#### Feedback

The user is presented with loading bars as the login operation is processed, while the friend's closeness evaluation is in progress and while the page is reloaded after pressing the "Start over" button. On the login screen, the user receives error messages if the login fails or (s)he forgets to accept the Terms of Service and Privacy Policy. On the main screen, the user receives error messages if there are not at least 2 friends selected, which is a requirement for the algorithm to work. The user is also prompted to confirm the logout procedure and receives a confirmation message after the logout has completed successfully.

#### Constraint

The user is not permitted to login unless (s)he accepts the Terms of Service and Privacy Policy, a legal requirement to use the app. The user is not allowed to login unless (s)he provides a valid Facebook username and password. The user is not allowed to proceed with the evaluation unless (s)he selects at least 2 friends, a requirement for the evaluation algorithm to work. A confirmation message is displayed before the logout procedure is initiated.

#### Mapping

The logout button embeds the "Power off" icon, which establish a relationship between the Logout option of our software and the real-world symbol for turning off electronic appliances. The "Start over" button also embeds the "Repeat" icon, present on the repeat button of the most music players.

## **Consistency**

Internal consistency is ensured by the fact that the look and feel is consistent across the application. The labels of all the buttons have the same font size. The same applies for the error messages and confirmation messages. The branding (logo) remains in the same place across the different screens of the application, and the Terms and Conditions and Privacy Policy links remain in the footer.

## **Affordance**

The users can easily recognise the actions that can be performed. When a button is clicked it appears to have been depressed, due to the shading. The icons are presented next to the button labels, making all the actions more easily recognisable (e.g. the power off icon of the “Logout” button, and the repeat icon of the “Start over” button).

## **Predictability**

The iFriends interface enables the user to predict what will happen if certain actions are taken. For example, a confirmation window is displayed after the “Logout” button is clicked, a feature that matches the behavior of most applications. Additionally, a friend disappears from the selected list, when the user tries to remove a specific friend.

## **Synthesizability**

The buttons in the iFriends application are clearly labelled. Contextual descriptions are placed on each web page to assist the user. Some interfaces in our system deliver immediate honesty (i.e. when you add a person to the people you wish to compare to and searching for friends), while other interfaces deliver eventual honesty (i.e. such as the as the final results).

## **Familiarity**

The elements of the iFriends interface are placed in positions that the user should be familiar with from past interactions from other applications. For example, the Logout button and the name of the user are displayed in the upper-right corner of the screen, are similar to Facebook’s web page layout. Other components are made to feel familiar, such as the remove person button, the search bar and the logout buttons. Another way in which this was achieved was via using domain specific language such as “login/logout”.

## **Justification for implementation choices**

A lot of the implementation choices we made were based on the research we had gathered about measuring relationship strength. Of particular use was the paper “Predicting Tie Strength with Social Media” by Eric Gilbert et al. The paper had the results of numerous tests done to see how various factors influenced relationship strength. This information was very valuable as it was quantifiable and applicable data that applied directly to social media sites like Facebook.

In particular it helped with the development of the algorithm as it showed how much things like messages exchanged between two people affected their relationship etc. Using all the various factors from the paper we created multiple functions each one using the Facebook API to find

out certain information such as messages exchanged between two individuals or pictures which they both were in etc. However due to time constraints not all of these were implemented, with us implementing just some of the most crucial ones. This is something that we must continue to work on to complete the project, and thereby fine tune the algorithm further.

We initially had plans on implementing a machine learning algorithm to help with calculating the tie strength. However this would have been too complex and computationally expensive, not to mention that the time required to implement it was better used for more core functionality. Instead we make a profile for each user when evaluating their friends, taking an average of the user's activity and then comparing each friend's interactions with that profile. The results from this method are very similar and within acceptable degrees of accuracy but much more efficient.

Similarly upon coming to the implementation we decided not to create an admin section. The reason for this is that the admin section was to handle the settings for API result paging, since the FB class that handled the API data did not implement paging, we have not included it.

### **Assumptions Log**

Through out the project the assumption log has been kept in a dynamic state. The assumption log was used to capture, document, and track the assumptions throughout the project. Usually each assumption should be assigned someone who is responsible for following up and validating the assumption, however in this case the responsibility fell to the team as a whole due to the group's decision to go for an ego and leaderless team structure. Each assumption was categorised and an action was assigned to validate/tackle that assumption.

The following assumptions were made throughout the lifecycle of this project via a webform made in Google Drive, this provided a nice interface to submit assumptions with very little effort (screenshot of form below).

## Assumption Form

**\*Required**

**Category \***  
Identify the area impacted by this assumption.

☐ Analysis

☐ Design

☐ Implementation

☐ Testing

☐ Maintenance

☐ Other:

**Assumption \***  
Describe the assumption being made.

**Time Stamp:** 13/11/2014 17:22:28

**Assumption:** The client will not change their requirements.

**Category:** Analysis

**Action:** Identify all the project requirements.

The assumption was closed upon the completion of the requirements presentation at the end of the first stage of development. The final requirements document made use of MoSCoW to reach a common understanding with the client on the importance they place on the delivery of each requirement, whether it was functional or nonfunctional.

**Time Stamp:** 13/11/2014 17:27:50

**Assumption:** The team will produce an adequately in-depth analysis before beginning to design software.

**Category:** Analysis

**Action:** Identify work required, then complete said work.

The assumption was closed upon the completion of the requirements presentation at the end of the first stage of development. The team researched the various different aspects of the project. Research papers relating to social networking were studied to establish the scope of the project and to develop an algorithm which would work friendship strength. As the project was based on social media, various social networking sites were examined for their main functions and how their API worked which would be used to pull data. The project being web based the hardware requirements of building such a system were also researched. Due to the longevity of the project different management techniques were studied and scrutinised.



**Time Stamp:** 14/11/2014 13:12:39

**Assumption:** The client will not change their requirements.

**Category:** Design

**Action:** Negotiate system requirements with client, alter project scheduling to accommodate changes.

The team scheduled meeting on a regular basis with the client in order to discuss project and getting feedback. This meant sometimes the systems requirements changed (such as only using Facebook as the social networking site) and this led to some alteration in the timeplan of the project.

**Time Stamp:** 14/11/2014 13:18:01

**Assumption:** User information we want will be available.

**Category:** Design

**Action:** Build in contingency to allow algorithm to work with the information missing.

The user and their friend may not have much interaction or information about themselves on Facebook, we built in contingency strategies so that when information was missing we could still calculate a friendship value.

**Time Stamp:** 21/11/2014 14:36:04

**Assumption:** Client doesn't change the requirements, after the requirements presentation.

**Category:** Design

**Action:** Regress back to analysis stage and research into solution again, based of new changes.

Immediately following the presentation , the team met with the client in order to receive feedback on the presentation. The client decided no additional to the original requirements.

**Time Stamp:** 27/01/2015 14:19:53

**Assumption:** Team have enough programming experience to tackle any programming related issues.

**Category:** Implementation

**Action:** Allocate jobs to certain coders, based on the difficulty of the task, and their experience.

The coding team were in regular contact with each other during the implementation stage. If a member of the team was struggling with their associated task then team worked together in order to tackle the issue.

**Time Stamp:** 27/01/2015 14:22:20

**Assumption:** The design document shall be completed by specified deadline.

**Category:** Design

**Action:** Allocate more time to working on the project.

This was not an issue as the team met the deadline specified.

**Time Stamp:** 27/01/2015 14:24:08

**Assumption:** The design produced shall meet our client's expectations.

**Category:** Design

**Action:** Negotiate necessary changes, then make changes to design document.

Following the submission of the design report, the team arranged for a meeting with our client who pointed out a few things that were missing from our report (which were added to the final report) and the report structure, which didn't flow. The team allocated two members to work on the report.

**Time Stamp:** 24/03/2015 08:59:03

**Assumption:** Can access enough user information to create accurate model of friendship.

**Category:** Implementation

**Action:** Change access more user information via other sources.

To calculate the most accurate friendship values we had to access as much information as possible. If we couldn't find enough information on facebook would have had to use a different social media site.

**Time Stamp:** 24/03/2015 09:00:30

**Assumption:** Team members shall complete their work within the given deadlines.

**Category:** Implementation

**Action:** Remind individual of work or re-delegate task.

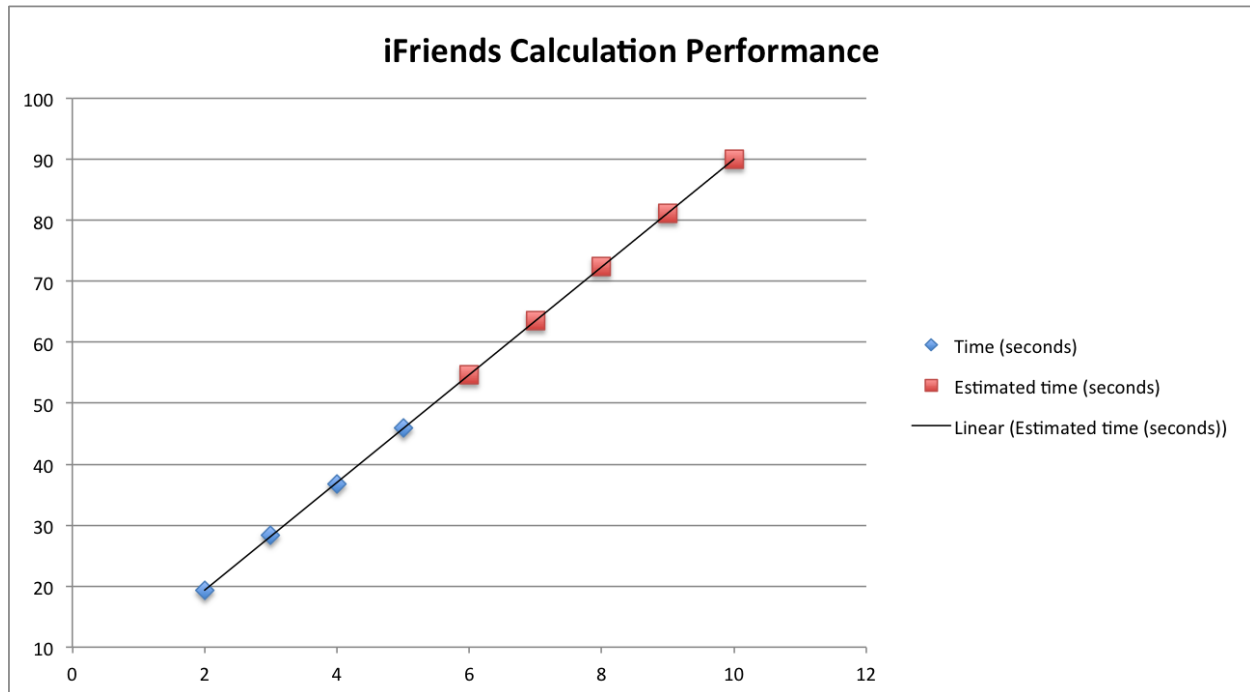
This wasn't a big issue for the team as the majority of the work was completed on time and if somebody was falling they were given the opportunity to finish their work.

## Time Complexity

It was important to test the performance of the system, and understand the time it would take to make the evaluation of a user's friends. In order to establish the time taken tests were run evaluating the friendship between the user and a number of friends as shown in the graph below. After evaluating the time taken to evaluate up to four friends I theorised that it takes approximately 9 seconds per friend for the system to finish its evaluation.

After running further tests to see the result of this theory it was found that each additional

friend added 9 seconds plus or minus 1 second to the evaluation. This puts the system as having  $O(n)$  time complexity.



*The blue nodes represent results measured by running the latest version of the program. The red nodes represent estimated times for additional friends. The trendline is a linear estimate based on the data recorded.*

### Software strengths & limitations

In terms of hardware limitations they are varied, and very much depend on the server the application is run on, which I cannot predict. We have attempted to minimise software limitations by not using pThreads and simply sticking to standard web server features, reducing complications during the installation phase.

The reduce complications with updating the Facebook API query code has been segmented into its own class that makes the standardised FB queries align to the current Graph API v2.2 standard. This means that keeping the website up-to-date with Facebook's latest supported API versions is significantly simplified, with the maintainer only having to look at the single class, rather than trawl through substantial amounts of code.

Adding and removing friendship variables has been simplified, due to the intentional low coupling in the program, removing variables requires as little as removing a single string from an array. To add an additional friendship variable a function must be written, added to the run method sequence and also added to the string array, though this may sound like a lot of steps, these steps are very simple and require little technical knowledge of the program.

As discussed earlier we have found that Facebook have implemented some limitations to prevent their service being abused, our system is also subject to these limits, which have been mentioned earlier in this report. [\[Facebook Query Limits\]](#)

The program very predictable calculation times, though the time required may not be particularly low since we do all the calculations on the server the results are consistent. This is especially significant with the every increasing popularity of the mobile platform, users on mobile devices or low power pcs should expect to receive the same experience as desktop/laptop users.

Currently the diagram can only display so many people, for large number of friends the diagram will have nodes overlap. To overcome this a solution like ordering the user by their score and displaying the nodes in a spiral shape could help, or even simpler just making the node images smaller.

The current algorithm does not use machine learning to identify trend in certain data sets, this could have helped keep the application relevant and accurate in the future without any hard coded modifications to offset variables (specified in the algorithm of the design report).

## Requirements

### MoSCoW

Here we have listed the MoSCoW requirements specified in the Requirements Presentation we did in the Autumn Semester. For each requirement we have briefly acknowledged whether or not we achieved the requirement.

### Must Have

- Connected to at least one social media site
  - Connects to Facebook
- Generating of all “friends” connected to the user
  - Generates results of all “friends” connected to the user that also have used the app/accepted the application’s permissions
- Group connections into different levels
  - Uses sliding scale from 0 to 1 to represent different friendship strengths
- Measure metrics using internal and external methods
  - Algorithm uses internal information, such as Facebook inbox messages, and external information such as user location to estimate friendship strength.
- Data extracted must relate to the chosen user
  - The application only requests personal information that relates to the selected friends. It does also access benign, publicly available information from facebook like longitude and latitude of a specific hometown.
- User permissions for data retrieval, processing and storage.

- Along with the Facebook login screen requesting permission to access specific data, a privacy policy and terms of service have also been added to the website.

#### Should Have

- Choose which social network
  - Not implemented
- Useable by more than one person
  - Once approved, anyone will be able to login to the app and look and their calculated friendship strengths
- Be given permission to access user's data
  - The Facebook login screen asks the user to give us permission to access their data.
- Manually change generated data
  - Not implemented
- Store queried data for future use
  - Not implemented
- Use data for commercial purposes
  - A Business Plan has been made, but not implemented
- Data presented in an easy to interpret format
  - Test users found the diagram clear, and easy to interpret
- Prevent personal data access without permission
  - Achieved by low persistence of data, personal information is removed from the server's memory as soon as the session has ended.

#### Could Have

- Automatic ordering of greeting cards
  - Not implemented, could partner with Moonpig to either link to their website or access their API.
- A way to export the final data
  - Not implemented, though the user can simply print the webpage, if the wish to keep a copy.
- Incorporation of Social Media sharing features
  - Not implemented
- Connect with multiple Social Media sites
  - Not implemented
- User specified friend classifications
  - Not implemented
- Non-binary friend classifications
  - Uses sliding scale from 0 to 1 to represent different friendship strengths.
- Recognise "fake" family members
  - Program ignores user defined information about relationships
- Multi-platform access

- The program may be accessed on any device with a web browser that supports: HTML5, cookies, JavaScript, and webpage redirecting.
- Generate data of “friends” who are connected on more than one site
  - Not implemented

### Success Criteria

In the Requirements Presentation we also specified a success criteria, which we planned to use to procure whether this project was an overall success or a failure. We have managed to achieve 5 of the 11 criteria, which would indicate a failed project, but crucial features like providing the ability to add extra features and fix code allows these extra features to be added with ease.

#### User

See who my closest friends are across multiple social media sites

**Failed** - The application does not support the use of multiple social media sites, just Facebook.

See how close I am with certain friends

**Achieved** - We present a diagram containing the friends the user wishes to analyse on the options screen.

See how many connections I have

**Achieved** - Friends that score 1 have no connection to the user except being friends on Facebook.

Choose which social media sites I use

**Failed** - The user can only use Facebook, there is no other selectable option.

Grant permission

**Achieved** - The application is granted inherent permission to use the user's data by Facebook when the user logs into application and grants access to specific data.

Have the ability to manually change the results

**Failed** - No such functionality has been implemented.

Share the results with friends

**Failed** - The feature has not been implemented as intended, though user can share their results by saving/printing the web page.

See where the people are who I interact with

**Failed** - Since we only allow users to look at Facebook results all interactions are from Facebook, which is not the feature we intended to produce.

#### Admin

Fix bugs

**Achieved** - Code has been well documented to allow for bug fixing to be easier.

#### Add new features

**Achieved** - Due to the object oriented approach we have used in this project procedures like implementing additional social media sites, additional variables, storing results, or including a share button have been simplified.

#### Monitor the consumer base

**Failed** - This feature has not been implemented into the application yet.

## Business Plan

**Mission Statement:** iFriends will be constructing a piece of software that will attract Facebook users to enter personal details and will calculate friendship strengths with friends that they have selected.

The proposed business plan will consist of advertisement, which will be the main focus throughout the business plan, along side another proposed idea which would include date reminders for the users, and include a link to an external website to buy gifts/cards to send. If we were to implement the advertisement scheme this means it would appear at different stages of the software and would encourage the user to click on links that will direct them to external websites. This is a necessary step as this will provide a source of income which will help provide future services for users that visit the software. Currently the software aims to provide a graphical representation of how close the user is to their selected friends list. Currently there are no current costs which will allow us as a group to spend more on the suggested business plans.

There are various ways in which the advertisements will be able to be implemented. Using AdMob will provide advertisement on all platforms and offers many mobile banners and text ads that can suit any developer's needs. This runs on a PPC (Pay Per Click) basis. This will make sure that we are charged accordingly. After research the average cost will be 72p/ click which will generate a healthy amount of income for our business.

Another proposed idea would be that the application could be used as a reminder for the users, which would send them notifications for important dates such as Birthdays/Anniversaries. These could be set up to alert the users of important dates, of the people within their 'inner circle' of friends. With regards to how to relate this to a business, this would present the user with one of two tasks:

- Write on Wall
- Send an E-Card (these can be done on other websites, and then posted on to people's walls).

These could be simple tasks that could be integrated into the app and could help prompt users to wish someone a Happy Birthday/Anniversary.

If we were to develop the application further developed it could then be used to:

- Send an actual card and send a gift.

This then opens up a new scenario. It would then link up to either:

- Link up to a website such as Moonpig.
- Create our own service (At the moment this wouldn't seem viable, however this would depend on how large the business would grow.)

This would create revenue through the use of external businesses, paying a specified amount to the business for the link to their website.

As regards to costing, this would be tricky to define. For the first two aspects: Write on a Wall, or Send an E-Card, this would mostly be free. It would not actually cost the project any money, but could cost the customer i.e. If they wanted to buy a specific E-Card template. It would however take time to integrate these, therefore this would then cost the project in time. With regards to the third aspect, this would not cost the business but would actually create revenue. If we were to link our webpage to moonpig, we could be able to create a contract with moonpig, and include a link within the application for users to go onto their website to buy cards/gifts to send.

The fourth aspect would be the most costly, as this would mean that we would actually have to set up our own business, which as known includes many costs. This would be the most riskiest venture, and without specific research on how many users of the app there are, and how many users of the app would use the service this would be a large risk to implement without adequate facts and figures.

## Issues Considered

### Legal

#### Code Ownership

The author of the PHP code is the one who has originally developed the script. This means that when the work has been handed over into the group environment they still have the rights to the work and they are responsible if anything was to occur with their chosen script.

Through the development process, and when the end product is completed with all PHP code being integrated, the holder of the work is usually integrated and the group is entitled to the software package and may be able to edit certain work that they may not have developed.

The group is regarded of having the copyright licensing when the work is handed over to the client. As mentioned in the previous report it is encouraged for the licence to be called "Cardiff University Group 6©."



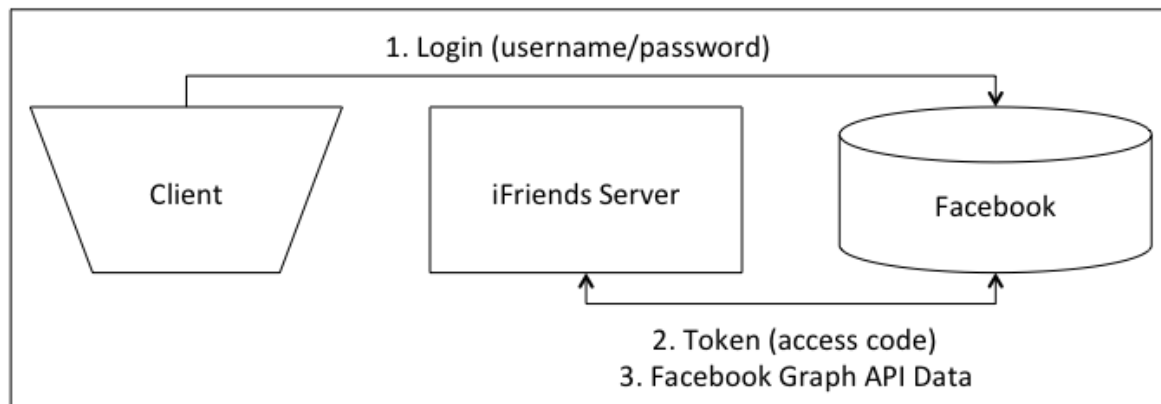
### Software Licensing

A considerable amount of research has been taken to decide what software licensing to use with our software. As a group we discussed what we wanted from our license and we researched to what software closely relates to our preferences. GNU GPL (General Public Licence) and MIT. They both have their advantages but unlike MIT, GPL doesn't have any protection and allows users to take our work and use it freely for any work, basically allowing us to be a third party software development group. MIT will allow people to use our software for other development processes but they must agree to use our copyright and disclose that they have used our work. This way our name will be on all software that has been developed with help from our code. This finding has meant that we have opted to use MIT as our software licensing and will be distributed to all work included in the software.

### Facebook Platform Policy

Facebook have some very strict platform policies<sup>[1]</sup>, which we have attempted to adhere to. The section that concerns us the most is chapter 3, "Protect Data" similar to the data protection act this dictates that we must not distribute user data, or handle Facebook integral information such as their Facebook password. As well as stress requirements for the way we handle information Facebook also provides guidelines on how they wish the interface to appear, buttons (like the log in/out buttons) must be clearly labeled, so the user knows what the button does.

### Data Protection



*All the user's information is contained within the box during the user's session (data persists in system until user exits their web browser).*

To conform with the Data Protection act and Facebook Platform Policy we have been very careful with the distribution of personal data. All sensitive information about the user and their friends is strictly sent between the iFriends server and Facebook, sensitive data does not move outside of this area - illustrated above. Once the client's session has expired the all sensitive data is removed from the iFriends server. Data that isn't sensitive such as FB user-ids and friendship values are sent between the server and client, if there was a malicious

attack, such as a rogue proxy the between the two ends, no sensitive information could be captured.

### **Ethical**

When finalising the application we have stuck to goals that have been previously mentioned in past reports. Initially we laid out that the aim of the application was to access and analyse user's personal details which will be processed. We have carried this objective efficiently making sure that ethics have been taken into consideration. A consent has been clearly messaged to the users before they sign into the software making sure that they are aware of what information the software will use. It has been hard for the group to implement a secure server to store the data so it has been finalised that no data will be stored and instead cache is used and will be erased after the user's session has expired.

The group has been successful in making sure that, only images used are the profile pictures. This therefore has reduced the risk of copyright as the user and Facebook will give the necessary permission. Overall, we have given close attention to the ethical problems and have made sure we've adjusted our software to prevent problems that we may face and are happy with the final outcome.

### **Social**

Working in large groups, there is always a risk that problems may arise with certain group members because of clashes either with work or personalities. Throughout the year there has been very few and no more than ordinary. But differences are that problems have been sorted almost instantly and have not affected the overall project in the slightest. The group has remained professional throughout. As all group members originate from Cardiff University in the same age range we believe this has prevented many social issues that are common amongst society.

### **Professional Issues**

The group has been professional throughout the software development cycle. The agile method has helped as it has meant the group has had to keep on task making sure work was completed to set deadlines. In every meeting one group member made every effort to ensure that content and issues were being spoke about meaning that all group meetings were effective and efficient. All members have acted professional in all stages of the development and has meant that our final product is well presented and works to the client's needs.

### **Conclusion**

According to our success criteria we have failed to produce the program we designed with the client. The program just meets the "Must have" requirements of the MoSCoW, not all the functions have been implemented, and significant sections of the system are missing - not great.

Though we did not complete the project we have managed to jump several of the more significant hurdles that we expected to run into. The first being finding an approach to

calculating ties strength, through a bit of research we have been able to tackle this issue. We then ran into the issue of gathering the information, we managed to identify the Facebook API as being an excellent source of social information. We have managed to understand how to use the Facebook API, and understand it well enough to produce code without the access to the API. Then when it came to collating the code, we had managed to predict challenges and design the system to accommodate for them.

From a technical standpoint this project has further extended the problem solving and programming skills of its members. When assigning code tasks, team members would be told the input, the output, and what the code should do. It was up to the programmer to figure out how to do manipulate the data they were provided to do what was required of the code. When creating their solution many team members did extra research in the PHP documentation to try identify the best method.

When looking at the other side of the project, we have learnt that meetings can be very useful for teamwork, and found that group productivity was significantly diminished over the holidays. We also realise the importance of report writing, they are useful tools for fully thinking through a solution's design or for reflecting on a project, identifying where there were problems and figuring out solutions retrospectively for the next project.

Past the system developed we (the team) have managed to develop ourselves, this project will help individuals retrospectively identify themselves in one or more of Belbin's team roles. The project has given all members an understanding the process of a large scale team project and some of the problems associated with such projects. We are a diverse team, with different backgrounds and a wide range of personalities, this project has caused us to gain insight into working closely with others with minimal conflict. We understand that for a project to be successful, all team members must be working synchronously, while maximising the man hours and reducing temporal deficits caused by task dependencies.

As a team we are proud of the product we created, a highly polished application designed with expansion in mind, and though we didn't complete it on time, there are already some team members who are interested in finishing the program off - over the summer holidays.

## **Future Work**

In terms of future work this program can go very far, excluding the work we would have to do to meet the success criteria, there are a lot of features that could be added or tweaked.

The primary focus in terms of completing the application would be to finish writing the FB API class, the class is still missing important functionality, like getting user photo information. To improve the speed of the FB class we would have liked to have completely implemented our caching solution - we had designed a cache that could have potentially reduced the number of Facebook requests made by approximately 30%.

For the algorithm we would like to implement a machine learning system, this could help us produce even more accurate results over time. This would have involved adding a few more objects to our code, and would make adding extra friendship variables much more complicated.

A small feature that could help improve the popularity of the application would be a share button that allowed the user to show their friends how well their friendships scored. In a discussion we decided the best way to do this would be to convert the diagram we present to the user into an image that could be posted to the user's Facebook timeline.

After we did a performance analysis of the program, we realised that we could easily predict the amount of time the algorithm would approximately take, in a future revision we would like to see an indicator of the time the app will take to complete the calculation. This would help improve the visibility of the system's current state, as while it's calculating the results the user doesn't know how long it will take or whether it has broken.

## References

- [1] Facebook Platform Policy (<https://developers.facebook.com/policy/>) - last updated 25 March 2015
- [2] Pew Research Centre  
(<http://www.pewresearch.org/fact-tank/2014/02/03/6-new-facts-about-facebook/>) - last updated 3 February 2014
- [3] Gilbert, E & Karahalios, K 2009, 'Predicting Tie Strength With Social Media', In *Proc. of CHI*
- [4] Social News Daily  
(<http://socialnewsdaily.com/35782/app-reveals-which-of-your-facebook-friends-makes-you-the-happiest/>) -last accessed 13th April 2015
- [5] Krakjoe, pThreads, (<https://github.com/krakjoe/pthreads>) - last updated January 2015
- [6] Tom's Planner, 2015, 'Tom's Planner: Online Gantt Chart - Project Planning software', (<http://www.tomsplanner.com/>)
- [7] Facebook Brand, Using Facebook Brand Assets, (<https://www.facebookbrand.com/>) - 29 April 2015

## Appended Zip File Information

The appended zip file contains all significant code mentioned in the report, including all the code for the final solution.

### Zip Structure:

- Work.zip
  - Implemented Code
    - iFriends Website
  - Other Code
    - Admin Section
    - Prototypes - Ben
  - Programming Outlines
    - Front-End Layer
    - Week 5
    - Week 6
    - Week 7
    - Week 8
    - Week 9
    - Back-End Layer

## Appendix

### Timings

#### Implementation Work

- Week 5
  - Abdu (Week 7)
  - Andrei (Week 6)
  - Ben (Week 5)
  - Karl (Week 5)
  - Rob (Week 5)
  - Rhian (Week 5)
  - Sam (Week 5)
  - Zain (Week 7)
- Week 6
  - Abdu (Week 7)
  - Andrei (Week 7)
  - Ben (Week 6)
  - Karl
  - Rob (Week 9)
  - Zain (Week 6)
- Week 7
  - Abdu (Week 7)
  - Andrei (Week 8)
  - Ben (Week 7)
  - Karl
  - Rob (Week 9)
  - Zain (Week 8)
- Week 8
  - Adbu (Week 9)
  - Andrei (Front-End)
  - Ben (Back-End)
  - Karl (Back-End)
  - Rob (Week 9)
  - Zain (Week 9)
- Week 9
  - Abdu (Week 9)
  - Andrei (Front-End)
  - Ben (Back-End)
  - Karl (Back-End)
  - Rob (Week 9)
  - Zain (Task 1: Week 9, Task 2: Week 9)

## Team Organisation

- Week 5
  - Andrei - Set up ifriends.website/set up production server
  - Ben - Created git
- Week 6
  - Rhian - Set up meeting
  - Sam - Recorded and uploaded minutes
- Week 7
  - Rhian - Set up meeting, recorded meeting
  - Sam - Software Licenses Appropriate
- Week 8
  - Ben - Setup a test server
- Week 9
  - Andrei & Roberto - Set up privacy policy, and terms of service
  - Ben - Restart test server (x1000)

## Documentation Work

(Work on “to do” bullet points at top of document)

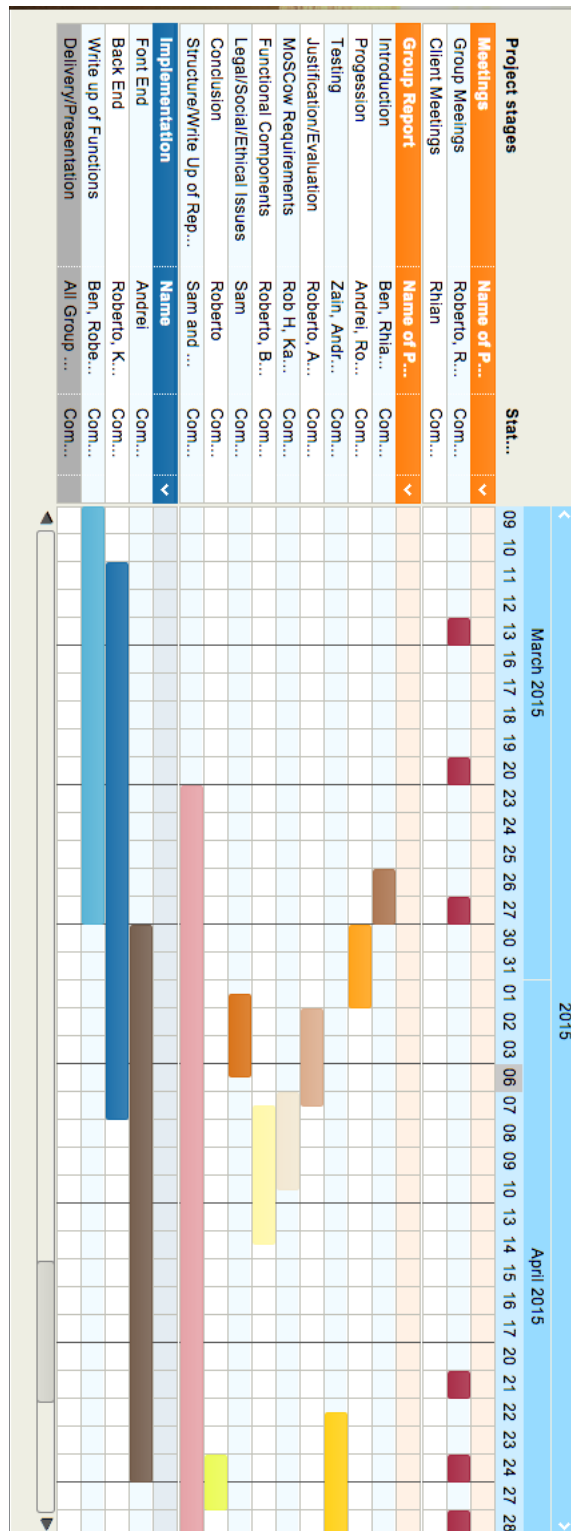
- Week 6
  - Rhian
  - Sam
- Week 7
  - Rhian
  - Sam
- Week 8
  - Rhian
  - Sam
- Week 9
  - Rhian
  - Sam
- Week 10
  - Abdu - Justify use of journal entry
  - Andrei & Zain - Justify design choices for User Interface
  - Ben - Discuss issues using pThreads & post work on paging inbox results
  - Karl - Describe group work progression in report
  - Rhian - Organise meeting with Matt Morgan to demonstrate work on Friday 1<sup>st</sup> May/Business Plan
  - Rob - Write up bullet points in Overall System Design section
  - Sam - Work on Legal/Ethical section/ Business Plan
  - Zain - Work on testing, also discuss assumptions & choices made
- Week 11
  - Ben, Rob, Zain - Introduction

- Rhian & Sam are on the business plan.
- Karl - analyse algorithm speed



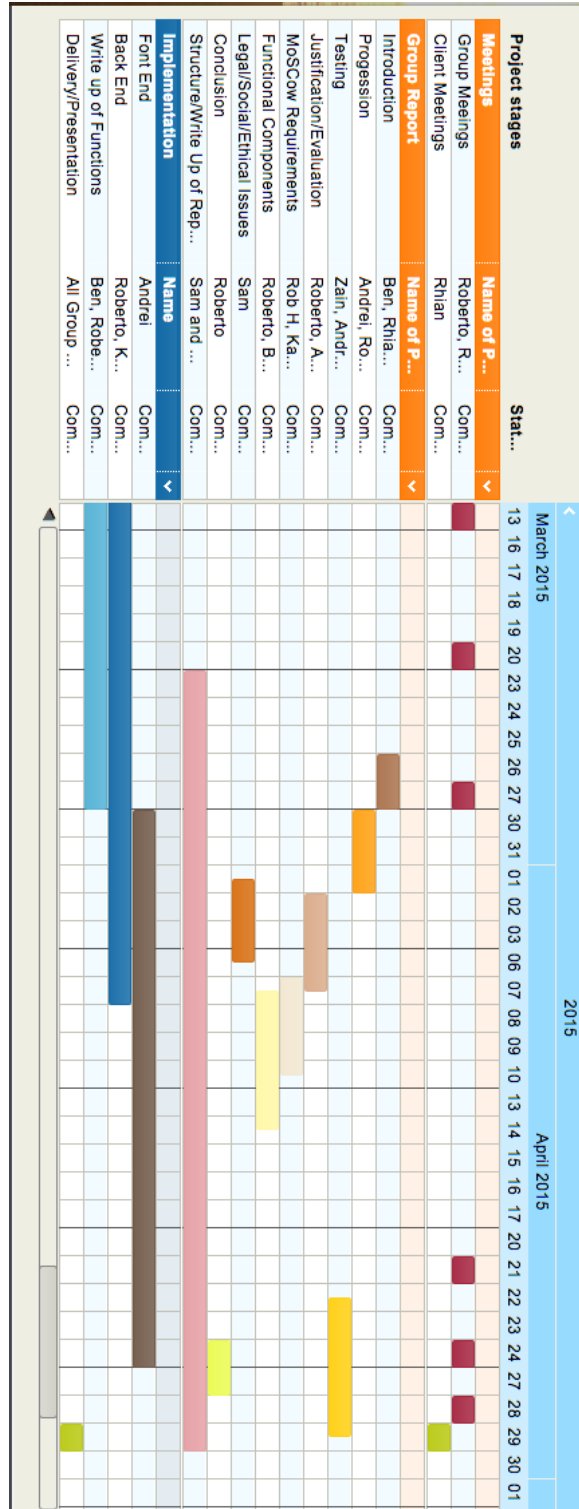
## Gantt Chart

### Part 1



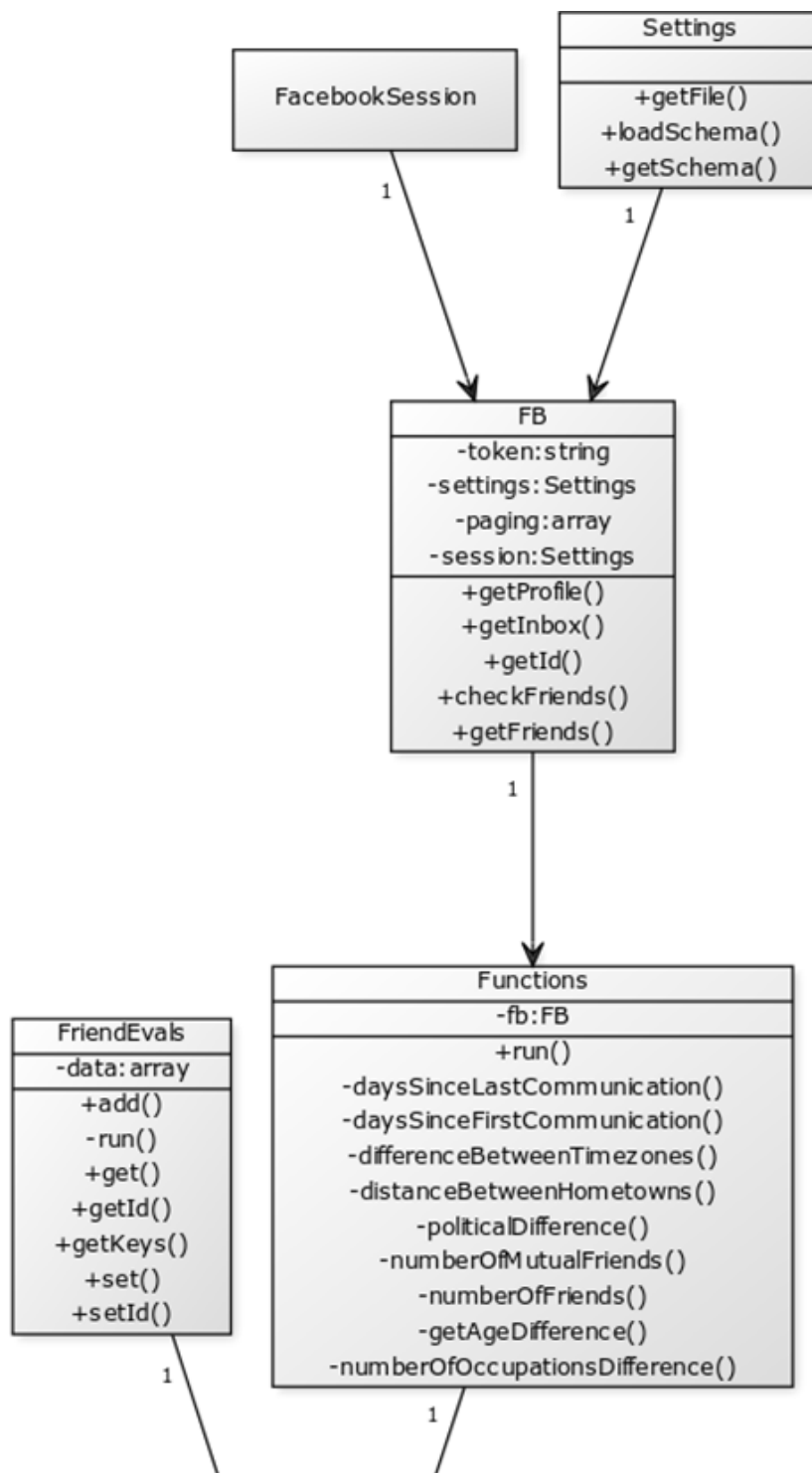
## Part 2

*Note: the design Gantt chart was only just too wide to fit so this screenshot is the similar to the first, but with the scroll bar moved slightly right.*



## Class Diagram

### Part 1



## Part 2

